Conceptes bàsics de Laravel

0- Namespace
1-Models i Migrations
2-Controllers
3-Views. Blades i Blade Templates
4-Routing
5- Request & Responses
6-Middleware
7- Seeders i Factories

Treballant amb Routing. Routing i Breeze

1- El concepte de **Routing** consisteix en redirigir la petició realitzada des de l'aplicació client cap a la funció del controller adequada en funció de:

- El mètode HTTP utilitzat (GET, HEAD, POST, PUT, PATCH, DELETE)
- L'adreça URL del recurs solicitat

2- Necessitarem una redirecció o ruta cap a:

- **create** i **store** per poder crear nous registres.
- **edit** i **update** i poder modificar registres a partir d'un criteri de selecció.
- **index** per poder veure tots els registres o veure una pàgina d'inici de l'aplicació.
- **show** per poder veure un registre a partir d'un criteri de selecció.
- **destroy** per poder esborrar un registre a partir d'un criteri de selecció.

3- Les rutes de:

- **create**, **edit**, **index** i **show** estaran associades al mètode GET (o HEAD)
- **store** estarà associat al mètode POST
- **update** al mètode PUT (o PATCH)
- **destroy** al mètode DELETE

4- Utilitzant les convencions de Laravel, les rutes per accedir als diferents mètode de la classe **ControladorTreballador** considerant que per accedir a aquesta part de l'aplicació s'utilitzarà la ruta **trebs** (He utilitzat aquest nom però podia haver utilitzat qualsevol altre) seran les següents:

Mètode	URI	Acció
GET/HEAD	trebs	index
GET/HEAD	trebs/create	create
POST	trebs	store
GET/HEAD	trebs/{treb}	show
GET/HEAD	trebs/{treb}/edit	edit
PUT/PATCH	trebs/{treb}	update
DELETE	trebs/{treb}	destroy

Què significa exactament aquesta taula?. Tres exemples per intentar entendre el seu significat:

- Si fem una petició a **http://<ip>:<port>/trebs/create** utilitzant el mètode HTTP **GET** llavors s'executarà el mètode **create** de la classe **ControladorTreballadors**. Aquest mètode hauria de retornar-nos la vista amb el formulari per introduir dades per crear un nou registre dins de la taula **treballadors**.
- Si feun una petició a **http://<ip>:<port>/trebs** utilitzant el mètode HTTP **POST**, llavor s'executarà el mètode **store** que crearà el nou registre dins de la taula **treballadors**.
- Si feun una petició a **http://<ip>:<port>/trebs{2}** utilitzant el mètode HTTP **DELETE**, llavors s'executarà el mètode **destroy** que esborrorà el registre identificat amb el número **2** dins de la taula **treballadors**. L'identificador de registre normalment serà el valor que està emmagatzemant a la clau primària. Així en el nostre cas, s'esborrarà el registre de taula **treballadors** amb el valor de **tid** (recordeu que a la taula **treballadors**, la clau primària és diu **tid** i és de tipus **intege**r) igual a **2**.

5- Com varem veure a la sessió 3, per registrar i fer in disponibles aquestes rutes simplement hem d'obrir **routes/web.php** i afegir la següent línia:

```
Route::resource('trebs', ControladorEmpleat::class);
```

i també ens cal afegir a la secció dels use la línia:

```
use App\Http\Controllers\ControladorTreballador;
```

6- Si a més a més, volem disponibles aquestes rutes només si ens hem validat i hem accedir al nostre **dashboard**, llavors haurem de crear una agrupació de rutes utilitzant **group**, controlades pel middleware d'autenticació creat quan varem afegir Breeze. En aquest cas, el fitxer **routes/web.php** hauria de tenir aquest codi:

```
<?php

use App\Http\Controllers\ProfileController;

use Illuminate\Support\Facades\Route;

use App\Http\Controllers\ControladorTreballador;

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

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

Route::group(['middleware' => 'auth'], function(){
        Route::get('/dashboard', function () {
                return view('dashboard');
        })->name('dashboard');

        Route::resource('trebs', ControladorTreballador::class);
});

Route::middleware('auth')->group(function () {

        Route::get('/profile', [ProfileController::class, 'edit'])->name('profile.edit');

        Route::patch('/profile', [ProfileController::class, 'update'])->name('profile.update');

        Route::delete('/profile', [ProfileController::class, 'destroy'])->name('profile.destroy');

});

require __DIR__.'/auth.php';
```

D'acord amb el codi indicat, si volem accedir a **http://<ip>:<port>/trebs** amb **GET** per veure una taula amb totes les dades de **treballadors** llavors:

- Podrem veure la taula si ens hem validat des de la pàgina de **login**
- Si no ens hem validat veurem que la pàgina de login serà mostrada per validar-nos.

Vinculant rutes, vistes, model i els mètodes create i store del controlador

NOTA IMPORTANT: A la **sessió 3** ja varem veure com vincular el mètode **index** de la classe de tipus controlador **ControladorTreballador** amb la vista **llista.blade.php**, la classe de tipus model **Treballador** i la ruta **http://<ip o nom>:<port>/trebs utilitzant mètode GET**. Ara veurem com podem desenvolupar vistes, rutes i els mètodes **create** i **store** del controlador.

1- Anem a afegir el codi dels mètodes **create** i **store** de **ControladorTreballadors**:

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

public function store(Request $request)
{
	$nouTreballador = $request->validate([
		'nom' => 'required',
		'cognoms' => 'required',
		'nif' => 'required',
		'data_naixement' => 'required',
		'sexe' => 'required',
		'adressa' => 'required',
		'tlf_fixe' => 'required',
		'tlf_mobil' => 'required',
		'email' => 'required',
		'treball_distancia' => 'required',
		'tipus_contracte' => 'required',
		'data_contractacio' => 'required',
		'categoria' => 'required',
		'nom_feina' => 'required',
		'sou' => 'required'
	]);
	$treballador = Treballador::create($nouTreballador);
	return view('dashboard');
}
```

D'acord amb aquests codis:

- Si accedim a **http://<ip o nom>:<port>/trebs/create** utilitzant el mètode **GET** llavors s'executarà el codi del mètode **create** de **ControladorTreballador**. La funció **view('crea')** retornarà el formulari **crea.blade.php** que s'hauria d'afegir a **resources/views**.
- En canvi, si accedim a **http://<ip o nom>:<port>/trebs** utilitzant el mètode **POST** llavors s'executarà el mètode **store** de **ControladorTreballador** i ens retornarà al nostre **dashboard**.

2- Crea dins de **resources/views** un fitxer de nom **crea.blade.php** tingui el següent codi que pots descarregar d'aquí. Bàsicament, és el formulari per omplir i enviar les dades del registre que volem crear dins de la taula i enviar les dades per introduir-les a la taul executant el mètode **store**. Comprova que:

- A la línia 18 el mètode HTTP és POST
- A la línia 18 la ruta és trebs

i que per tant, la ruta creada ens farà arribar al codi del mètode **store** del controlador.

Afegeix al final un enllaç que ens permeti tornar al **dashboard** si no volem omplir dades. Afegeix al final del fitxer abans de **$endsection** les següents instruccions amb **HTML** i llenguatge **blade**:

```
<div class="p-6 bg-white border-b border-gray-200">
	<a href="{{ url('dashboard') }}">Torna al dashboard<a/>
</div>
```

3- Modificarem **dashboard.blade.php** per tenir els següents enllaços:

- Un enllaç al formulari d'entrada de dades. Canvia la línia **12** (a on pots llegir *You're logged in*) per la següent instrucció **HTML** utilitzant a més a més llenguatge **blade**:

```
<a href="{{ url('trebs/create') }}">Crea un nou registre<a/>
```

- Un enllaç al formulari de visualització de la base de daes. Abans de la línia **11**, afegeix les següents instruccions amb **HTML** i llenguatge **blade**:

```
<div class="p-6 bg-white border-b border-gray-200">
      <a href="{{ url('trebs') }}">Mostra dades de la taula treballadors<a/>
</div>
```

4- També modificarem **llista.blade.php** per tenir un enllaç que ens permeti tornar al dashboard un cop mostrades les dades de la taula. Afegeix al final del fitxer aban de **$endsection** les següents instruccions amb **HTML** i llenguatge **blade**:

```
<div class="p-6 bg-white border-b border-gray-200">
      <a href="{{ url('dashboard') }}">Torna al dashboard<a/>
</div>
```

i a més a més, per entendre millor la informació mostrada farem que a la **línia 41**, el codi HTML sigui aquest:

```
<td>{{$treb->treball_distancia == "1" ? 'Sí':'No'}}</td>
```

5- Per millorar la presentació de les dades, modifica disseny.blade.php. Fes que la **línia 11** tingui el següent codi:

```
<div class="container-fluid" >
```

6- I ara finalment, comprovarem si podem accedir a l'aplicació, visualitzar les opcions del **Dashboard** i utilitzar:

- La pàgina de visualització de totes les dades de la taula **treballadors**.
- El formulari de creació d'un registre, enviar les dades, i insertar-les en la taula **treballadors**.
- Si inserim un registre ens torna al dashboard.
- No podem accedir a aquestes pàgines si no ens hem validat amb la pàgina de **login**.

Per fer aquest comprovació, seguirem aquests passos:

- Trobarem l'adreça IP de la màquina virtual executant: **ip a**
- Posarem en marxa el servidor intern que ens proporciona **laravel** executant dins del directori **empresa** l'ordre: **php artisan serve --host=0.0.0.0 --port=8000**
- Comprovarem que el servidor comença a funcionar escoltant pel port **8000/tcp**.
- Des de la màquina física accedirem a l'adreça IP i port del servidor de la màquina virtual.
- Crearem una nova entrada dins de la taula **treballadors** de la base de dades **empresa** i tornarem al **dashboard.**
- Visualitzarem la taula, comprovarem que s'ha afegit la nova entrada i podem tornar al dashboard.

Vinculant rutes, vistes, model i el mètode destroy del controlador. Utilitzant Spoffing pel mètode HTTP DELETE.

1- Abans de res, farem alguns canvis a dashboard per fer més fàcil l'utilització de l'aplicació. Canviarem la **línia 12** de manera que tingui aquest codi:

```
<a href="{{ url('trebs') }}">Treballadors: Visualitza, actualitza i esborra registres<a/>
```

i canviarem la **línia 15** de manera que tingui aquest codi:

```
<a href="{{ url('trebs/create') }}">Treballadors: Crea un nou treballador<a/>
```

2- Ara anem afegir una nova columna a la taula mostrada per **llista.blade.php** que no estarà associada a cap dada de la taula **treballadors** sino que servirà per tenir un espai per afegir uns botons per les **accions** *esborra*, *actualitza* i *mostra* que afegirem més endavant. Entre les línies **24** i **25** de **llista.blade.php** afegeix:

```
<td>Accions sobre la taula</td>
```

Ara, la línia **25** tindrà el nou codi HTML.

3- I ara, afegirem un botó d'accío amb l'etique ***"Esborra"*** que quan es faci clic a sobre, cridi al mètode **destroy()** de **ControladorTreballador**. Entre les línies **47** i **48** de **llista.blade.php** afegeix:

```
<td class="text-left">
 <form action="{{ route('trebs.destroy', $treb->tid)}}" method="post" style="display: inline-block">
  @csrf
  @method('DELETE')
  <button class="btn btn-danger btn-sm" type="submit">
   Esborra
  </button>
 </form>
</td>
```

Dins d'aquest codi són d'especial interés els següents aspectes:

- Laravel permet afegir molt fàcilment spoofing method dins d'un formulari HTML pels mètodes **DELETE**, **PUT** i **PATCH** amb la directiva **@method**.
- La directiva **@csrf** permet afegir protecció contra atacs CSRF dins d'un **form** d'HTML.
- El helper **route** fa una feina similar al helper url per amb alguna diferència. En el nostre cas, indicarem que:
  - S'ha d'executar el mètode **destroy()** associat a la ruta **trebs**. Com hem vist a la **sessió 12 - punt 5** (a la pàgina 10), la ruta **trebs** està associada a **ControladorTreballador**, de manera que s'executarà el mètode **destroy()** de **ControladorTreballador**.
  - Al mètode **destroy** li passarem el paràmetre **$treb→tid** que identificarà el treballador que volem esborrar.

4.- Ja tenim la vista. Ara anem al controlador. El nou codi del mètode **destroy()** de **ControladorTreballador** serà:

```
public function destroy($tid)
{
       $treballador = Treballador::findOrFail($tid)->delete();
       return view('dashboard');
}
```

Aquest codi trobarà el registre amb `findOrFail($tid)` i l'identificador que li passem com a paràmetre, i l'esborrarà amb `delete()`. A continuació, tornarà a la vista **dashboard**.

Vinculant rutes, vistes, model i els mètodes update i edit del controlador. Utilitzant Spoffing pel mètode HTTP PUT.

1- Anem a fer canvis en el codi de l'aplicació per poder afegir l'opció d'**editar** i **actualizar un registre** de la taula **treballadors**. Començarem per afegir un botó d'acció ***"Edita"*** a **llista.blade.php** que quan es faci clic a sobre cridi al mètode **edit()** de **ControladorTreballador**. Entre les línies **48** i **49** de **llista.blade.php** afegeix:

```
 <a href="{{ route('trebs.edit', $treb->tid)}}" class="btn btn-primary btn-sm">Edita</a>
```

2- El mètode **edit** del **Controlador:**

- Crida al **Model** per recuperar de la taula les dades del registre identificat per **$treb→tid** que volem editar i actualitzar i desar-les dins d'una variable.
- Envia la variable amb les dades recuperades a un **Vista** que té un formulari per poder veure les dades i canviar-les. En aquest cas el formulari es diu **actualitza.blade.php**.
- El seu codi serà aquest:

```
public function edit($tid)
{
    $dades_treballador = Treballador::findOrFail($tid);
    return view('actualitza',compact('dades_treballador'));
}
```

3- El codi de la Vista **actualitza.blade.php** es po trobar aquí. Podem veure que:

- És un formulari que recull les dades enviades pel Controlador i les mostra. És especialment interessant veure com es recuperen les dades dels **select** a les línies **36 a 42**, **63 a 69** i **70 a 79**.
- El formulari utilitza **spoofing method** pel mètode **PATCH** (o PUT) i protecció **CSRF**.
- Crida al mètode **update()** de **ControladorTreballador** passant l'identificador **tid** del registre a actualitzar.

4- Ara ja només cal afegir el codi del mètode **update()** de **ControladorTreballador**. El codi serà el següent:

```
public function update(Request $request, $tid)
{
        $noves_dades_treballador = $request->validate([
                'nom' => 'required',
                'cognoms' => 'required',
                'nif' => 'required',
                'data_naixement' => 'required',
                'sexe' => 'required',
                'adressa' => 'required',
                'tlf_fixe' => 'required',
                'tlf_mobil' => 'required',
                'email' => 'required',
                'treball_distancia' => 'required',
                'tipus_contracte' => 'required',
                'data_contractacio' => 'required',
                'categoria' => 'required',
                'nom_feina' => 'required',
                'sou' => 'required'
        ]);
        Treballador::findOrFail($tid)->update($noves_dades_treballador);
        return view('dashboard');
}
```

Deixant de banda que al principi posem la llista de camps que és obligatori tenin omplerts en el formulari, la línia important és aquesta:

```
Treballador::findOrFail($tid)→update($noves_dades_treballador);
```

Aquesta línia busca el registre i quan el troba l'actualitza amb les noves dades enviades des del formulari.

Vinculant rutes, vistes, model i els mètodes show per visualitzar un registre específic.

1- Anem a fer canvis en el codi de l'aplicació per poder afegir l'opció de **visualitzar** dades d'un registre concret de la taula **treballadors**. Començarem per afegir un botó d'acció ***"Mostra"*** a **llista.blade.php** que quan es faci clic a sobre cridi al mètode **show()** de **ControladorTreballador**. Entre les línies **56** i **57** de **llista.blade.php** afegeix:

```
<a href="{{ route('trebs.show', $treb->tid)}}" class="btn btn-info btn-sm">Mostra</a>
```

2- El mètode **show** del **Controlador:**

- Crida al **Model** per recuperar de la taula les dades del registre identificat per **$treb→tid** que visualitzar dins d'una variable.
- Envia la variable amb les dades recuperades a un **Vista** que permet veure les dades. En aquest cas el formulari es diu **mostra.blade.php**.
- El seu codi serà aquest:

```
public function show($tid)
{
    $dades_treballador = Treballador::findOrFail($tid);
    return view('mostra',compact('dades_treballador'));
}
```

3- El codi de la Vista **mostra.blade.php** es pot trobar aquí. Podem veure que és un formulari que simplement recull les dades enviades pel **Controlador** i les mostra.

Creació de llista2.blade.php per fer l'aplicació més utilitzable

1- Crea un fitxer de nom **llista2.blade.php** amb següent codi que trobaràs aquí dins de **resouces/views**.

2- Modifica el mètode **index** de la classe **ControladorTreballador** i fes que el seu nou codi sigui:

```
/**
* Display a listing of the resource.
*/
public function index()
{
	$dades_treballadors = Treballador::all();
	return view('llista2', compact('dades_treballadors'));
	// Recollirà totes les entrades de la taula treballadors i les desarà dins d'una
	//variable de nom $dades_treballadors
	//Cridara a la vista llista.blade.php que es trobarà a resouces/views per
	//mostrar les dades dels treballadors
	//The compact() function creates an array from variables and their values.
}
```

3- Comprova que ara la vista de la llista d'usuaris mostra dades bàsiques dels treballador (nom i cognom) i si volem veure totes les dades del treballador hem de premer el botó **Mostra**.

Creant una pàgina d'error per error d'accés a la base de dades

1- Afegeix a **resources/views** un fitxer de vistes de nom **errobd.blade.php** amb el següent codi:

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
	<head>
		<meta charset="utf-8">
		<meta name="viewport" content="width=device-width, initial-scale=1">
		<title>Error d'accés a la BD</title>
	</head>
	<body>
		<h1>Atenció!!!!</h1>
		<p>Error tipus: "SQLSTATE[HY000] [2002] Connection refused"</p>
		<p>
			Comprova que:
			<ol>
				<li>El servidor MySQL està en marxa</li>
				<li>L'adreça IP i/o nom del host són correctes</li>
				<li>El port del servidor és correcte</li>
				<li>El nom d'usuari i contrasenyes són correctes</li>
				<li>El fitxer de configuració de Laravel és correcte</li>
			</ol>
		</p>
	</body>
</html>
```

2- Modifica **bootstrap**/**app.php**. Dins del seu codi, afegeix:

- A la secció dels **use** afegeix:

```
use Illuminate\Database\QueryException;
use Illuminate\Http\Response;
use Illuminate\Support\Facades\View;
```

- El codi de la secció **->withExceptions** serà aquest:

```
->withExceptions(function (Exceptions $exceptions) {
        $exceptions->renderable(function(QueryException $exception) {
                if ($exception->getCode()=='2002'){
                        return new Response(View::make('errorbd'));
                };
        });
})->create();
```

3- En aquest exemple:

- **->withExceptions** gestiona les excepcions (com per exemple, les excepcions que es produeixen quan hi ha un error d'accés a la base de dades) i permeten crear logs de les excepcions i renderitzar pàgines de resposta.
- El mètode **renderable()** mostra la vista **error.blade.php** si es produeix un error de connexió a la base de dades.
- La vista **errobd.blade.php** crear una pàgina mostrant un missatge d'error i d'aquesta manera es pot personalitzar el comportament de gestió d'errors.

4- Atura el servidor **MySQL** i comprova que es mostra **error.blade.php** quan s'intenta establir una connexió a l'aplicació.