Conceptes bàsics de Laravel

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

Treballant amb migrations i afegint el bastiment Laravel Breeze - Sessió 9

1- Comprova que dins del directori **empleats** que en el moment de la instal·lació del framework de Laravel s'ha creat un fitxer **.gitignore** i que el fitxer **.env** està inclòs dins de la llista dels fitxers que no es trobaran en seguiment pel sistema i per tant tampoc es pujaran a **Github**.

2- Dins del directori **empresa** crea un dipòsit local **git** i fes que **main** sigui el nom de la branca principal:

```
git  init
git branch -m main
```

3- Fes un un **add** del projecte i un **commit** amb el missatge *"Commit 1 del projecte empresa"*.

4- Crea un dipòsit **privat** de **Github** de nom **empresa**. Sincronitza el dipòsit local amb el dipòsit remot.

5- artisan és el nom d'una eina que s'inclou amb cada instal·lació del framework de **Laravel** i que necessita l'interpret de **php** per ser interpretada i executada. Artisan permet fàcilment, entre altres opcions, crear **taules, models**, **controladors** i **vistes**. L'ordre **artisan** es trobarà dins del directori **empresa** que és on s'ha instal·lat el framework de **Laravel**.

6- Les **migrations** de **Laravel:**

- Permeten crear, esborrar, reanomenar o canviar l'estructura de les taules d'una base de dades sense haver d'executar ordres de MySQL entrant a la base de dades o crear un script amb ordres SQL per fer aquesta feina.
- Les Migrations es defineixen dins de fitxers que es troben al subdirectori **database/migrations**.
- Haurem d'utilitza l'eina **artisan** per executar les migrations que s'aniran definint dins del projecte.
- Per defecte ja existeixen unes migrations definides en el moment de crear un projecte de **laravel**.
- Executarem les **migrations** amb l'ordre:
-

  ```
  php artisan migrate
  ```

  i comprova que s'han creat les taules: **failed_jobs**, **migrations**, **password_resets**, **personal_access_tokens** i **users**.

7- Laravel té un bastiment per poder fer fàcil el procés de registre i autenticació d'usuaris per l'aplicació. Entre d'altres té el paquet **Laravel Breeze** que d'acord amb la documentació: "*Laravel Breeze is a simple, minimal implementation of all of Laravel's authentication features, including login, registration, password reset, email verification, and password confirmation".* Per instal·lar Laravel Breeze, dins del directori **empresa** executa:

```
composer require laravel/breeze --dev
npm install && npm run dev
```

Aquest bastiment aprofitarà les taules creades amb la primera migration realitzada per poder emmagatzemar usuaris i les seves contrasenyes. També ens proporciona unes vistes per poder fer feines típiques d'autenticació.

8- Ara creaem una fitxer de migrations per crear la taula **empleats** dins del directori de **migrations**. Volem els següent camps: **id** (identificador), **nom** de tipus text, **telefon** de tipus text, **email** de tipus text i **timestamps**. Primer haurem de crear el fitxer de migrations executant:

php artisan make:migration create_empleats_table --create=empleats

i farem que el contingut sigui aquest:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateEmpleatsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('empleats', function (Blueprint $table) {
            $table->id();
            $table->string('nom');
            $table->string('email');
            $table->string('telefon');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('empleats');
    }
}
```

9- Crea la taula **empleats** executant des del directori empresa:

php artisan migrate

A continuació, entra dins de MySQL i executa:

use empresa;
show tables;
describe empleats;

i comprova que s'ha creat la taula correctament.

10- Consulta la taula **migrations** de la base de dades empresa executant:

select * from migrations;

i comprova que el resultat similar a:

```
MariaDB [empresa]> select * from migrations;
+----+-------------------------------------------------------+-------+
| id | migration                                             | batch |
+----+-------------------------------------------------------+-------+
| 18 | 2014_10_12_000000_create_users_table                  |     1 |
| 19 | 2014_10_12_100000_create_password_resets_table        |     1 |
| 20 | 2019_08_19_000000_create_failed_jobs_table            |     1 |
| 21 | 2019_12_14_000001_create_personal_access_tokens_table |     1 |
| 22 | 2023_02_18_113949_create_empleats_table               |     2 |
+----+-------------------------------------------------------+-------+
5 rows in set (0.014 sec)
```

Això és un **log** de les **migrations** realitzades fins ara i que a més a més, ens permet controlar la manera de fer **rollbacks**. Un **rollback** ens permet **revertir** la darrera, les N darreres o totes les **migrations** realitzades.

Surt del client **mysql** i reverteix la darrera **migration** executant des de la carpeta **empresa**:

```
php artisan migrate:rollback
```

A continuació entra a empresa/database/migrations i esborra el fitxer PHP **2023_02_17_xxxxxx_create_empleats_table.php** (el valor xxxxxx pot canviar en funció de l'hora de creació del fitxer).

Hem vist doncs, com podem revertir i eliminar els efectes d'una **migration**.

11- Ara anem a crear una taula que en comptes d'empleats es dirà **treballadors** i tindrà més camps. Executa des de la carpeta **empresa**:

```
php artisan make:migration crea_taula_treballadors --create=treballadors
```

de manera que a **empresa/database/migrations** es crearà el fitxer: **2023_mm_dd_hhmmss_crea_taula_treballadors.php** (a on **mm** és el mes, **dd** és el dia i **hhmmss** és l'hora de creació del fixer). A continuació, obre el fitxer PHP creat i fes que la estructura de camps de la taula treballadors sigui aquesta:

```
$table->id('tid');
$table->string('nom');
$table->string('cognoms');
$table->string('nif')->unique();
$table->date('data_naixement');
$table->char('sexe',1);
$table->string('adressa');
$table->integer('tlf_fixe');
$table->integer('tlf_mobil');
$table->string('email');
$table->binary('fotografia')->nullable();
$table->boolean('treball_distancia');
$table->enum('tipus_contracte',['temporal','indefinit','en formació','en pràctiques']);
$table->date('data_contractacio');
$table->tinyinteger('categoria');
$table->string('nom_feina',50);
$table->float('sou');
$table->timestamps();
```

NOTA:

- Tipus de columnes: https://laravel.com/docs/10.x/migrations#available-column-types
- Modificadors de columnes: https://laravel.com/docs/10.x/migrations#column-modifiers
- Tipus d'index: https://laravel.com/docs/10.x/migrations#available-index-types

Ara, des del directori empresa executa:

```
php artisan migrate
```

A continuació, entra dins de MySQL i executa:

```
use empresa;
show tables;
describe treballadors;
```

i comprova que s'ha creat la taula correctament.

12- Si volem modificar ara un camp de la taula, per exemple, fent que el camp `'nom_feina'` tingui **30** caràcters en comptes de **50**, haurem de:

- Editar **2023_mm_dd_hhmmss_crea_taula_treballadors.php**
- Modificar el camp `'nom_feina' ==> $table→string('nom_feina',30);`

A continuació, executarem dins del directori **empresa**:

php artisan migrate:refresh

Després, amb l'ordre **describe treballadors;** dins de **MySQL** comprovarem que s'ha modificat el camp.

13- Així doncs, ara ja sabem:
- Fer migrations
- Revertir i eliminar migrations
- Treballar amb tipus de dades i els seus modificadors
- Modificar migrations

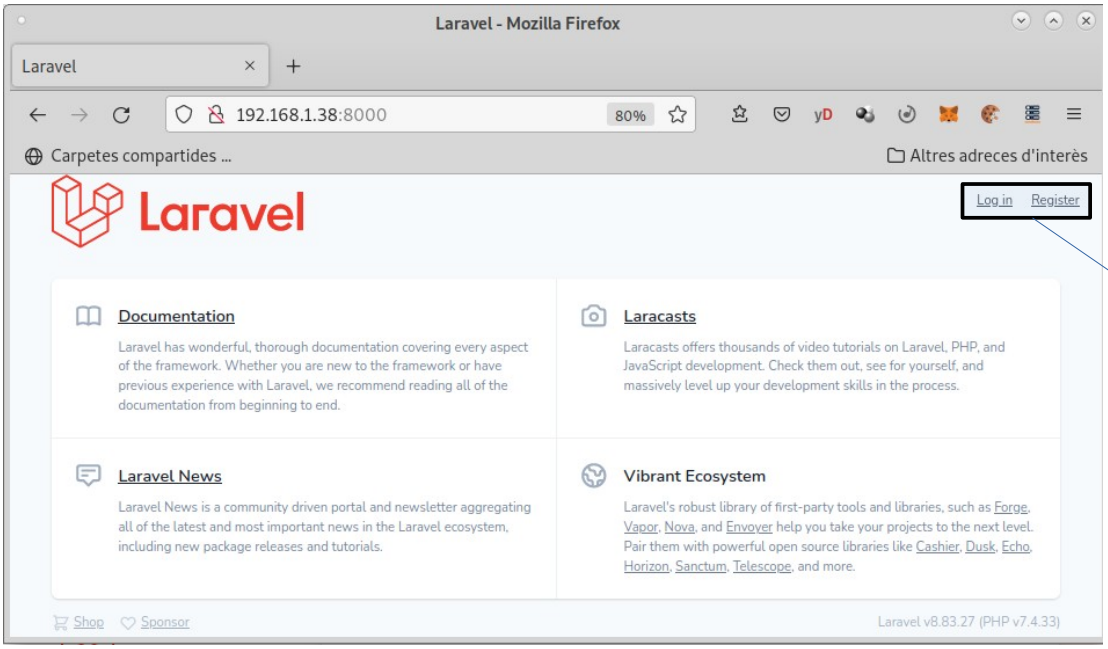
14- Per finalitzar aquesta sessió 9, anem a fer accessible el bastiment **Laravel Breeze** que hem instal·lat en el punt **7**. Des de dins el directori **empresa** executa:

php artisan breeze:install blade
npm install && npm run dev
php artisan migrate

15- Comprova com està la nostra aplicació:

- Troba l'adreça IP de la teva màquina virtual gestionada amb vagrant. Executa: **ip a**
- Executa des del directori **empresa**: **php artisan serve --host=0.0.0.0 --port=8000**

i ara, des de la teva màquina física accedeix a l'adreça IP de la màquina virtual i utilitzant el port **8000** i comprova que el resultat és aquest:



Noves opcions de Login i Registre ara disponibles.

16- Comprovacions:
- Accedeix a l'opció **Register** i comprova que pots enregistrar un nou usuari i que un cop registrat et porta al **Dashboard** de l'usuari.
- Comprova que pots fer un **logout** i tornes a la pàgina d'inici.
- Accedeix amb **Log in** i comprova que pots validar-te i accedeixes al **Dashboard** de l'usuari, i que també que pots tancar la sessió amb un **logout** i tornar a la pàgina inicial.
- Comprova que s'ha creat l'usuari a la taula **users** de la base de dades **empresa**. La contrasenya s'haurà desat en format **hash** i tindrem informació de la **data de creació**.

17- Si no es poden veure bé les pàgines de de registre, login i dashboard es per un petit problema de CSS amb la versió **1.10.0** de **Larabel Breeze**. Es pot solucionar fent un downgrade a la versió **1.9.4** anant al directori **empresa** i executant:

```
composer require laravel/breeze:1.9.4
php artisan breeze:install blade
```

i si es refreca la pàgines des del navegador haurem veurem correctament les pàgines de registre, login i dashboard.

18- Finalment, personalitzarem la pàgina d'inici de l'aplicació. Per fer això, primer obrirem **routes/web.php** i canviarem l'**enrutament** de **/** perquè obri un fitxer de nom i**nici.blade.php** que es trobarà a **resources/views**. És molt fàcil, el codi de les línies 16 a 18 de **web.php** serà:

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

Bàsicament, estem dient que enrutem **/** cap a **inici.blade.php**. No cal indicar l'extensió perquè la funció view() ja s'encarrega de buscar el fitxer dins de **resources/views**.

Després crearem el fitxer **inici.blade.php** dins de **resources/views** 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>Empresa</title>
	</head>
	<body>
		<div>Pàgina inicial de l'aplicació web Empresa</div>
		@if (Route::has('login'))
			@auth
				<a href="{{ url('/dashboard') }}">Dashboard</a>
			@else
				<a href="{{ route('login') }}">Log in</a><br>
				@if (Route::has('register'))
					<a href="{{ route('register') }}">Register</a><br>
				@endif
			@endauth
		@endif
	</body>
</html>
```

Refresca el navegador i comprova que ara la pàgina inicial ha estat personalitzada, tot i que es pot millorar afegint CSS.

Aquest darrer pas, perment introduir 2 conceptes:

- Enrutament a partit de la pàgina **web.php** de la carpeta **routes**.
- Els fitxers **blade** que són plantilles per poder fer la part de les **vistes** del patró **MVC**, que es troben a la carpeta **resources/views** i que tenen el seu propi llenguatge conegut com Blade templating language amb:
  - Directives tipus @if, @switch, @for, @foreach, @while
  - Directives d'autenticació: @auth → Comprova si un usuari ha estat o no autenticat
  - {{ }} que es l'equivalent a executar un echo de PHP `{{ route('register') }}` fa un echo de la ruta al blade de registre.

Treballant amb Models - Sessió 10

1- En el **patró de programació MVC** (Model-Vista-Controlador), el **Model** és la part de l'aplicació responsable de les interaccions amb les taules de les bases de dades i per tant, és la part encarregada de fer accions típiques d'INSERT, SELECT, UPDATE, DELETE sobre les taules. El **Model** mai s'encarrega d'accions com interactuar amb l'usuari o acceptar mètodes HTTP GET, POST, PUT o DELETE.

2- Laravel:

- Inclou un ORM (Object-Relational Mapper) anomenat Eloquent ORM.
- Eloquent ORM defineix una classe abstracta anomenada **Model** a partir de la qual, per herència, es poden crear noves classes que tinguin tots els atributs i mètodes necessàris per fer INSERT, SELECT, UPDATE i DELETE sobre les taules de la base de dades.
- Amb Eloquent ORM, totes i cadascuna de les taules de la base de dades de l'aplicació han d'estar associades al seu model propi, el qual ha de tenir una classe pròpia que hereta de **Model** i que tindrà tots els atributs i mètodes necessàris per fer INSERT, SELECT, UPDATE i DELETE sobre aquella taula en concret.
- Per tant, amb Eloquent ORM, per cada taula hem de crear un model propi que dins seu defineix una classe pròpia necessària per interactuar amb la taula.

3- Algunes convencions per defecte de Laravel importants a tenir en compte és la següent:

- Les taules s'escriuen en plural i totes les seves lletres estan en minúscula. Un exemple seria la taula **treballadors** que hem creat a la *sessió 9 - punt 11*.
- La classes tenen el mateix nom que la taula però la primera lletra s'escriu amb majúscules i s'escriu en singular. Així doncs, el nom de la classe del model associat a la taula treballador hauria de ser **Treballador.**
- El fitxers PHP amb els models de les taules es desen a la carpeta **app/Models** de l'aplicació.
- El fitxers PHP amb els models tenen el mateix nom que la classe, de manera que per la taula **treballadors** és crearà una classe de nom **Treballador** que hereta de **Model**, i un fitxer **app/Model/ Treballador.php**.

4- Dins del projecte **empresa**, per crear el fitxer del model amb la classe **Treballador** necessari per interactuar amb la taula **treballadors** seguint les convencions de Laravel, hem d'anar a la carpeta **empresa** i executar:

```
php artisan make:model Treballador
```

i en el cas de no haver creat prèviament la taula **treballadors**, es pot crear al mateix temps el fitxer del model i el fitxer de migrations executant:

```
php artisan make:model Treballador --migration
```

5- Comprova que s'ha creat el fitxer **app/Model/Treballador.php** i que dins té una classe de nom **Treballador** que hereta de **Model**.

6- Laravel assumeix per defecte que la taula té una clau primària de nom **id**. En el cas de que aquest no sigui el cas, s'haurà de canviar el nom de la clau. Hi ha un exemple de com fer aquest canvi aquí. És interessant llegir Eloquent Model Conventions per coneixer altres assumpcions que fa Laravel per defecte.

7- Els models tenen una sèrie d'atributs que es poden utilitzar per modificar el seu comportament per defecte. A la secció https://laravel.com/api/10.x/Illuminate/Database/Eloquent/Model.html pots trobar alguns d'aquest atributs i el seu significats. Alguns atributs interessants són:

- **$table** → Per canviar el nom de la taula associada al model
- **$primaryKey** → Per canviar el nom de la clau primària
- $incrementing → Per indicar si la clau primària és autoincremental(true) o no (false)
- **$keyType** → Tipus de la clau primària. Per exemple 'string' si és una cadena de caràcters.
- **$fillable** → És un array. Indica els camps de la taula que es poden omplir enviant una array amb les dades. En Laravel aquest procés és anomenat Mass assignment. Utilitzar aquest sistema fa que el procés d'omplir les dades sigui més eficient però pot ocasionar alguns problemes de seguretat. Amb l'opció $fillable indiquem els camps que es podem omplir fent Mass assigment. Bàsicament són els camps que l'usuari pot omplir amb un formulari. Els camps que no es trobin dins la lllista, no es poden omplir via Mass assignment, és a dir, no estaran en el formulari per l'usuari.

- **$guarded** → Camps que NO poden ser omplert via Mass assigment. Tots aquells camps que NO es trobin aquí dins, per defecte es poden omplir via Mass assignable, és a dir, poden estar disponibles en el formulari per l'usuari.

8- Així doncs, d'una manera molt bàsica, si per exemple volem que només els camps **nom**, **cognoms** i **nif** de la taula **treballadors** es pugui omplir amb les dades enviades des d'un formulari omplert per l'usuari, i si la clau primària és **nif** (que és un string i no és autoincremental) hauríem de modificar la nostra classe **Treballador** des del fitxer **app/Models/Treballador.php** i escrivint:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Treballador extends Model
{
	use HasFactory;
	protected $primaryKey = 'nif';
	public $incrementing = false;
	protected $keyType = 'string';
	protected $fillable = [
		'nom',
		'cognoms',
		'nif'
	];
}
```

Ara bé, en el nostre cas:

- La **clau primària** de la taula **treballadors** és el camp **tid**, que és és **integer** i **autoincremental**.
- Els camps a omplir són: **nom**, **cognoms**, **nif**, **data_naixement**, **sexe**, **adressa**, **tlf_fixe**, **tlf_mobil**, **email**, **fotografia**, t**reball_distancia**, **tipus_contracte**, **data_contractacio**, **categoria**, **nom_feina** i **sou**.

Per tant, un exemple de com podria ser la nostra classe **Treballador** dins del fitxer **app/Models/Treballador.php** seria la següent:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Treballador extends Model
{
	use HasFactory;
	protected $primaryKey = 'tid';
	protected $fillable = [
		'nom','cognoms','nif','data_naixement','sexe','adressa',
		'tlf_fixe','tlf_mobil','email','fotografia','treball_distancia','
		'tipus_contracte','data_contractacio','categoria','nom_feina','sou'
	];
}
```

9- Alguns mètodes interessants de la classe **Treballador** serien:

- **Treballador::all()** → Fa un **SELECT** de tots els registres de la taula **treballadors**.
- **Treballador::create($dades)** → Fa un **INSERT** i crea un nou registre en la taula **treballadors**.
- **Treballador::findOrFail($tid)** → Busca un registre a partir del valor de la clau primària **$tid**. Si el troba assigna el contingut del registre a una variable i si no el troba llença una excepció que pot ser capturada per mostrar un missatge de resposta del tipus **404**.
- **Treballador::where('nif', '=','10243876R')→first()** → Troba un registre. En aquest cas seria el registre amb el camp **nif** igual a **10243876R**. El resultat es pot assignar a una variable. Correspon a un **SELECT**.
- **Treballador::where('nif', '=','10243876R')→update($dades)** → Troba un registre i l'actualitza amb **$dades**. En aquest cas seria el registre amb el camp **nif** igual a **10243876R**. Correspon a fer un **UPDATE** a la base de dades.
- **Treballador::whereId($tid)→update($dades)** → Troba un registre amb la clau primària de valor **$tid** i l'actualitza amb **$dades**. Correspon també a fer un **UPDATE** a la base de dades. La variable **$dades** ha de ser un array associatiu amb els camps de la taula com a índex i els nous valors associats.
- També es pot fer un **UPDATE** així:
  $treballador=Treballador::where('nif','=','10243876R');
  $treballador→nif='98765432B';
  $treballador→save( );
- **Treballador::where('sou','=','2000')→update(['sou' => '2050']);** → Fa una actualització de multiples registres amb el nou valor de sou 2050 a partir dels registres que segueixen el criteri que el sou és 2000.
- **Treballador::findOrFail($tid)→delete( );** → Troba un registre amb la clau primària de valor **$tid** i l'esborra. Si no el troba llença una excepció. Correspon a fer un **DELETE** a la base de dades.
- **Treballador::where('nif', '=','10243876R')→delete( );**→ Troba un registre amb el nif indicat i l'esborra. Correspon també a fer un **DELETE** a la base de dades.

Treballant amb Controllers - Sessió 11

1- En el **patró de programació MVC** (Model-Vista-Controlador), un **Controlador (Controller):**

- Es la part de l'aplicació que rediregiex les peticions provinents de les Vistes cap el Model i mètode adequats tenint en compte el tipus de petició realitzada i rediregeix cap a les Vista adequada la resposta obtinguda.
- S'encarrega d'accions com interactuar amb l'usuari o acceptar mètodes HTTP GET, POST, PUT o DELETE i redireccionar-los cap el mètode del Model adequat.
- Mai s'encarrega d'interactuar directament amb les taules de les bases de dades i per tant, mai s'encarrega de fer INSERT, SELECT, UPDATE, DELETE sobre les taules.

2- Laravel:

- Defineix una classe abstracta anomenada **Controller** a partir de la qual, per herència, es poden crear noves classes de tipus **Controller**.
- Normalment crearem una classe de tipus Controller associada a una classe de tipus Model. Per exemple, per la classe **Treballador** tindrem associada un classe que es podria anomenar per exemple **ControladorTreballador** (de fet, no hi ha una convenció per el noms dels controladors).
- Els mètodes típics d'un controlador són:
  - **index** → S'utilitzaria normalment per redirigir peticions que demanen veure tots els continguts de la taula cap el mètode adequat del Model i retornar cap a la Vista adequada el resultat obtingut. Aquest mètode ha de cridar al mètode del Model que permet recollir totes els registres de la taula i les envia a la vista que els mostra amb el navegador. També es pot utilitzar per mostrar una pàgina d'inici de l'aplicació.
  - **show** → S'utilitzarà normalment per redirigir peticions que demanen veure el continguts de només un registre de la taula cap el mètode adequat del Model i retornar cap a la Vista adequada el resultat obtingut. Aquest mètode ha de cridar al mètode del Model que permet recollir un registre de la taula i les envia a la vista que els mostra amb el navegador.
  - **create** → S'utilitzar per demanar i mostrar el formulari necessari per afegir les dades d'un nou recurs (o sigui, un nou registre en la taula) que es vol crear. No crea el recurs, només es crida per mostrar el formulari.
  - **store** → S'utilitza per crear i emmagatzemar el recurs amb les dades proporcionades des del formulari mostrat amb **create**. És el mètode que crea el nou recurs (o sigui, un registre en la taula). Aquest mètode ha de cridar al mètode del Model que crea el recurs.

- **edit** → S'utilitzar per demanar i mostrar el formulari necessari per **actualitzar** les dades d'un recurs (o sigui, un nou registre en la taula) existent. No actualitza el recurs, només es crida per mostrar el formulari.
- **update** → S'utilitza per **actualitzar** el recurs amb les dades proporcionades des del formulari mostrat amb **edit**. És el mètode que **actualitza** el recurs (o sigui, un registre en la taula). Aquest mètode ha de cridar al mètode del Model que **actualitza** el recurs.
- **destroy** → S'utilitza per esborrar un recurs (o sigui, un registre en la taula).

- Els mètodes del controlador estan associats als següents mètodes HTTP:
  - GET/HEAD → index, show, edit, create
  - POST → store
  - PUT (i PATCH) → update
  - DELETE → destroy

5- Anem a crear el Controlador. Dins del projecte **empresa**, per crear **ControladorTreballador** executa:

php artisan make:controller ControladorTreballador --resource

i comprova que es crea el fitxer **app/ControllersControladorTreballador.php** dins del qual trobaràs la classe ControladorTreballador amb els mètodes **index**, **create**, **store**, **show**, **edit**, **update** i **destroy** ja preparats per ser desenvolupats.

Treballant amb Routing - Sessió 12

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- Per fer disponibles les rutes cap els controladors s'ha de **descomentar la línia 29** de **app/Providers/RouteServiceProvider.php**. És a dir:

```
protected $namespace = 'App\\Http\\Controllers';
```

NO ha d'estar comentada. En cas contrari, les rutes cap els controladors no podran ser utilitzades.

5- 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

Mètode	URI	Acció
GET/HEAD	trebs/{treb}/edit	edit
PUT/PATCH	trebs/{treb}	update
DELETE	trebs/{treb}	destroy

Això vol dir que una petició a **http://localhost/trebs/create** utilitzant el mètode HTTP **GET** farà que s'executi 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**. Però si el mètode HTTP és **POST**, llavor s'executar el mètode **store** que crearà el nou registre dins de la taula **treballadors**.

En aquest contexte **{treb}** és un identificador de registre que normalment serà el valor que està emmagatzemant a la clau primària. Així en el nostre cas, la petició a **http://localhost/trebs/{1}** amb el mètode **DELETE** farà que s'executi el mètode **destroy** de la classe **ControladorTreballadors**. Aquest mètode 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 **1**.

Per registrar i fer in disponibles aquestes rutes simplement hem d'obrir **routes/web.php** i afegir al final:

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

Ara bé, si volem tenir-les disponibles només si ens hem validat i accedir al **dashboard**, llavors haurem de fer un *Route* **group** (agrupació de rutes), i el fitxer **routes/web.php** hauria de tenir aquest codi:

```
<?php

/*
|--------------------------------------------------------------------------
| 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!
|
*/

use Illuminate\Support\Facades\Route;

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);
});

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