

Conceptes bàsics de Laravel

Conceptes bàsics	Passos a seguir per un projecte bàsic
1- Namespace 2-Models i Migrations 3-Controllers 4-Views 5- Blades i Blade Templates 6-Routing 7- Request & Responses 8-Middleware 9- Seeders i Factories	1- Creació de base de dades, usuari i permisos amb un script sql 2- Creació d'un projecte amb l'ordre laravel 3- Configuració del projecte modificant l'arxiu .env 4- Creació de taules del projecte: <ul style="list-style-type: none">• Creció dels fitxers de Migrations• Execució de les Migrations => Creació taules 5- [opcional] Afegiment de Middleware d'autenticació 6-Desenvolupament dels Models 7- Desenvolupament dels Controllers 8- Creació de Routes 9- Creació de Views (blades)

Treballant amb Models

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 que és una capa de software intermitja entre el codi PHP i les ordres SQL que actuen sobre la base de dades). Aquest ORM s'anomena [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 necessaris 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 propi model, el qual ha de tenir una classe pròpia que hereta de **Model** i que tindrà tots els atributs i mètodes necessaris 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- Respecte del nom de taules i models Laravel utilitza les següents convencions per defecte que són importants a tenir en compte quan es desenvolupa l'aplicació:

- 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 2 - Part 2 - punt c).
- 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- Laravel anomena també Model a un registre d'una taula d'una base de dades. Per tan hem de diferenciar quan parlem de les classes de tipus Model i d'un model, o sigui una entrada d'una taula.

5- 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

6- Comprova que s'ha creat dins de la carpeta del projecte **empresa** el fitxer de model **app/Models/Treballador.php** que dins té una classe de nom **Treballador** que hereta de **Model**.

7- 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 conèixer altres assumpcions que fa Laravel per defecte.

8- 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 amb la llista dels camps de la taula que es poden omplir a partir d'un array amb dades normalment rebut des d'un del formulari. En Laravel aquest procés és anomenat Mass assignment. Els camps que no es trobin dins la llista no haurien d'estar disponibles en el formulari per l'usuari
- **\$guarded** → Camps que NO poden ser omplerts via Mass assignment. Tots aquells camps que NO es trobin aquí dins, per defecte es poden omplir via Mass assignment, és a dir, poden estar disponibles en el formulari per l'usuari.

9- 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 **integer** i **autoincremental**.
- Els camps a omplir són: **nom**, **cognoms**, **nif**, **data_naixement**, **sexe**, **adressa**, **tlf_fixe**, **tlf_mobil**, **email**, **fotografia**, **treball_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'
    ];
}
```

10- 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 múltiples 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.

11- Per fer una prova que funcionen aquests mètodes farem una prova. Afegeix dins de **routes** → **web.php** el següent codi al final:

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

i després, crea dins de routes un fitxer de nom **treb.php** amb el següent codi:

```
<?php
```

```
use App\Models\Treballador;
```

```
Route::post('/noureg', function () {
    $dades= array("nom"=>"Jaume", "cognoms"=>"Pons", "nif"=>"46108810Q",
        "data_naixement"=>"2000-03-15", "sexe"=>"d",
        "adresa"=>"Carrer de València, 690, 1-2", "tlf_fixe"=>"934567890",
        "tlf_mobil"=>"666339900", "email"=>"jpons@gmail.com",
        "treball_distancia"=>true, "tipus_contracte"=>"temporal",
        "data_contractacio"=>"2018-03-15", "categoria"=>3,
        "nom_feina"=>"desenvolupador junior", "sou"=>1404,47);

    Treballador::create($dades);

    $dades= array("nom"=>"Anna", "cognoms"=>"Puig", "nif"=>"47108810P",
        "data_naixement"=>"2000-03-15", "sexe"=>"d",
        "adresa"=>"Carrer de València, 680, 2-2", "tlf_fixe"=>"934567899",
        "tlf_mobil"=>"666339911", "email"=>"apuig@gmail.com",
        "treball_distancia"=>false, "tipus_contracte"=>"temporal",
        "data_contractacio"=>"2021-03-15", "categoria"=>2,
        "nom_feina"=>"desenvolupador senior", "sou"=>1704,97);

    Treballador::create($dades);

});

Route::get('/mostrareg', function () {
    $nom="Jaume";
    $dades=Treballador::where("nom", "=", $nom)->first();
    echo $dades["cognoms"]."\n";
});

Route::delete('/delreg', function () {
    $nom="Jaume";
    Treballador::where("nom", "=", $nom)->delete();
});
```

12- Posa en marxa novament el servidor web intern de **Laravel** executant des del directori **empresa**:

```
php artisan serve --host=0.0.0.0 --port=8000
```

13- A continuació, amb l'ordre **curl** de la teva màquina física executa: **curl -X POST http://<ip_màquina_virtual>:8000/noureg** a on **<ip_màquina_virtual>** s'ha de canviar per l'adreça IP de la màquina virtual. Després, comprova que s'ha creat una entrada dins de la taula **treballadors**.

14- Si la resposta és una pàgina d'error amb el títol **"Page expired"** llavors, modifica el fitxer **bootstrap/app.php**. Fes que **withMiddleware** tingui aquest codi:

```
->withMiddleware(function (Middleware $middleware) {  
    //  
    $middleware->validateCsrfTokens(except: [  
        '/noureg',  
    ]);  
})
```

i torna a executar l'ordre **curl**. Si no dóna cap error, comprova que s'ha creat una entrada dins de la taula **treballadors**.

15- Amb l'ordre **curl** de la teva màquina física executa: **curl -X GET http://<ip_màquina_virtual>:8000/mostrareg**. A continuació comprova que es mostra el camp **cognom** d'un dels registres creats a l'apartat anterior.

16- Amb l'ordre **curl** de la teva màquina física executa: **curl -X DELETE http://<ip_màquina_virtual>:8000/delreg**. A continuació comprova que s'esborra l'entrada.

17- Si la resposta és una pàgina d'error amb el títol **"Page expired"** llavors, modifica el fitxer **bootstrap/app.php** **app/Middleware/VerifyCsrfToken.php**. Fes que **withMiddleware** tingui aquest codi:

```
->withMiddleware(function (Middleware $middleware) {  
    //  
    $middleware->validateCsrfTokens(except: [  
        '/noureg',  
        'delreg',  
    ]);  
})
```

18- Torna a crear el registre esborrat perquè ens caldrà més endavant.

Treballant amb Controllers. Començant a utilitzar Routes i vistes

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

- Es la part de l'aplicació que rediregeix les peticions provinents de les **Vistes** cap al **Model** adequat tenint en compte el tipus de petició realitzada i rediregeix cap a les **Vista** adequada la resposta obtinguda des del **Model**.
- Com que en un fitxer de **Model** normalment té una classe amb mètodes per fer INSERT, SELECT, etc.. dins d'una taula, el **Controlador** s'encarrega de seleccionar el mètode correcte en funció de la petició rebuda des de la **Vista**.
- Un **Controlador** accepta la recepció dels mètodes HTTP GET, POST, PUT o DELETE i redirecciona les peticions cap al mètode correcte dins de la classe adequada per gestionar la petició.
- 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 una classe que es podria anomenar per exemple **ControladorTreballador** (de fet, no hi ha una convenció per el noms dels controladors).
- Normalment, hi ha un fitxer PHP diferent per cada classe de tipus Controller.

3- Els mètodes típics d'un controlador són:

- **index** → S'utilitzaria normalment per redirigir peticions que demanen veure tots els continguts de la taula cap al 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 contingut de només un registre de la taula cap al 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
 - GET/HEAD/{url relativa addicional} → show, edit, create (recorda que create no crea registres)
 - POST → store
 - PUT (i PATCH) → update
 - DELETE → destroy

4- Abans de continuar, comentarem els darrers canvis que van fer perquè no segueixen el patró MVC:

- Comenta o esborra la línia `require __DIR__ . '/treb.php'`;
- Esborra (però no és realment necessari) el fitxer **treb.php** dins de **routes**.
- Comenta o esborra totes les línies noves que varem afegir a **withMiddleware** del fitxer **bootstrap/app.php**.

5- Ara crearem el controlador necessari per interactuar amb la classe del model **Treballador** i amb les vistes que farem més endavant. Dins del projecte **empresa**, executa aquesta ordre per crear el controlador:

```
php artisan make:controller ControladorTreballador --resource
```

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

6- A continuació, registrarem la ruta que apunta a la classe **ControladorTreballadors** dins de **routes/web.php** afegint les línies:

- Al final de la secció dels **use**:

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

per assegurar-nos que es pot accedir a la classe **ControladorTreballador** des de **web.php**. Aquesta serà ara la línia 5.

- A la secció de les **Route** afegeix:

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

a on **trebs** serà la ruta base necessària per arribar al controlador i els seus mètodes. És a dir, per arribar a un mètode de la classe **ControladorTreballador** s'utilitzarà la ruta base `http://<ip_equip>:<port>/trebs/`.

6- Ara indicarem el **Model** que volem utilitzar dins de **ControladorTreballadors.php**. A la secció dels **use** afegirem:

```
use App\Models\Treballador;
```

7- A continuació, podem començar a desenvolupar els mètodes del controlador. Per exemple, pel mètode **index** de la classe **ControladorTreballador** només ens cal el següent:

```
public function index()
{
    $dades_treballadors = Treballador::all();
    return view('llista', 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 resource/views per mostrar
    //les dades dels treballadors
    //The compact() function creates an array from variables and their values.
}
```

Imaginem que l'equip servidor tingui l'adreça IP **192.168.1.40** i el servidor web escoltats pel port **8000**. En aquest cas, el mètode **index** de la classe **ControladorTreballador** s'executaria quan féssim una petició amb el mètode **GET** a **http://192.168.1.38:8000/trebs**.

7- D'acord amb el codi del mètode **index.php** del punt anterior, ara hem de crear un fitxer de vistes de nom **llista.blade.php** dins de **resource/views**. A més a més, farem que aquesta vista tingui el següent codi per visualitzar totes les dades de tots els usuaris de la taula **treballadors**:

```
@extends('disseny')
@section('content')
<h1>Llista d'empleats</h1>
<div class="mt-5">
    <table class="table">
        <thead>
            <tr class="table-primary">
                <td>tid</td>
                <td>Nom</td>
                <td>Cognoms</td>
                <td>NIF</td>
                <td>Data de naixement</td>
                <td>Sexe</td>
                <td>Adreça</td>
                <td>Telèfon fixe</td>
                <td>Telèfon mòbil</td>
                <td>Email</td>
                <td>Fotografia</td>
                <td>Treball a distància</td>
                <td>Tipus de contracte</td>
                <td>Data de contractació</td>
                <td>Categoria</td>
                <td>Nom de la feina</td>
                <td>Sou</td>
            </tr>
        </thead>
        <tbody>
            @foreach($dades_treballadors as $treb)
            <tr>
                <td>{{ $treb->tid }}</td>
                <td>{{ $treb->nom }}</td>
                <td>{{ $treb->cognoms }}</td>
                <td>{{ $treb->nif }}</td>
                <td>{{ $treb->data_naixement }}</td>
                <td>{{ $treb->sexe }}</td>
                <td>{{ $treb->adresa }}</td>
                <td>{{ $treb->tlf_fixe }}</td>
                <td>{{ $treb->tlf_mobil }}</td>
            </tr>
            @endforeach
        </tbody>
    </table>
</div>
</section>
```

```

        <td>{{ $treb->email }}</td>
        <td>{{ $treb->fotografia }}</td>
        <td>{{ $treb->treball_distancia }}</td>
        <td>{{ $treb->tipus_contracte }}</td>
        <td>{{ $treb->data_contractacio }}</td>
        <td>{{ $treb->categoria }}</td>
        <td>{{ $treb->nom_feina }}</td>
        <td>{{ $treb->sou }}</td>
    </tr>
</tbody>
</table>
</div>
</endsection>

```

i com que aquest codi utilitza el contingut de **disseny.blade.php** com es veu a la primera línia, en el següent punt veurem com s'ha de crear i el seu contingut.

8- Crea el fitxer `disseny.blade.php` dins de `resources/views` i fes que el seu codi sigui el següent:

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>Projecte d'accés a BD MySQL amb Laravel 10</title>
        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css">
    </head>
    <body>
        <div class="container-fluid">
            @yield('content')
        </div>

        <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js"></script>
        <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"
            type="text/js"></script>
    </body>
</html>

```

9- Ara posa en marxa el servidor web intern de laravel executant des d'empresa**:**

php artisan web --host=0.0.0.0 --port=8000

i estableix una connexió des de la màquina física a l'adreça URL **`http://<ip_màquina_virtual>:8000/trebs`** (a on has de canviar **`<ip_màquina_virtual>`** per la seva adreça IP real). Comprova que el resultat es aquest:

Llista d'empleats

Id	Nom	Cognoms	NIF	Data de naixement	Sexe	Adreça	Telèfon fixe	Telèfon mòbil	Email	Fotografia	Treball a distància	Tipus de contracte	Data de contractació	Categoria	Nom de la feina	Sou
1	Jaume	Pons	46108810Q	2000-03-15	d	Carrer de València, 690, 1-2	934567890	666339900	jpons@gmail.com		1	temporal	2018-03-15	3	desenvolupador junior	1404
2	Anna	Pulg	47108810P	2000-03-15	d	Carrer de València, 680, 2-2	934567899	666339911	apulg@gmail.com		0	temporal	2021-03-15	2	desenvolupador senior	1704

