

Laravel

- Route :

1^{er} version :

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

2^{ème} version :

```
Route::get('users/{id?}', 'App\Http\Controllers\HomeController@home')->name('show');
```

?: optionnel

{id} : une route dynamique dépend du valeur id

- Controller :

>> php artisan make:controller HomeController

```
class HomeController extends Controller  
{  
    public function home($id=1){  
        return view('users.show', ['id'=>$id]);  
    }  
}
```

- View :

1) Layout.blade.php

```
2) <!DOCTYPE html>  
3) <html lang="en">  
4) <head>  
5)     <meta charset="UTF-8">  
6)     <meta name="viewport" content="width=device-width, initial-  
    scale=1.0">  
7)     <meta http-equiv="X-UA-Compatible" content="ie=edge">  
8)     <title>Document</title>  
9) </head>  
10) <body>  
11)  
12)     @yield('content')  
13)  
14) </body>  
15) </html>
```

2) user/show.blade.php

```
@extends('layout')

@section('content')
    <h1>Ayoub MNS, My id is {{$id}}</h1> //or {!! id !!}
@endsection
```

- Migration :

Il nous de manipuler les donnees de DB à travers laravel(créer des tables,modification...)Et **Elequent** qui transforme le code php en code sql

```
>> php artisan make:migration create_users_table
```

Nom du table :doit être minuscule et pluriels .

- Si on veut créer le model et la table en même temps :

```
>> php artisan make:model User -m
```

User : 1ere lettre en majuscule et singulier

- Si la table users est déjà existée :

```
>> php artisan make:model User
```

Pour manipuler les donnees on a besoin de Tinker (shell)

La table users(id,name,...)

- Set data

```
>> php artisan tinker
$user=new User //User :le Model
$user->id=1
$user->name= 'ayoub'
$user->save() // les donnees sont envoyés à la BD
```

- Get data

```
>> $Data=User ::cursor()
>>foreach($Data as $user) echo " {$user->id}\n {$user->name}\n" ;
```

Afficher le 1er user

```
>>User::find(1) //id=1 syntaxe Model ::find(id) ou Model ::find([id1,id2])
```

Afficher user qu'ont name='ayoub'

```
>>User ::where('name','ayoub')->get()
```

Afficher les names qui commence par 'mans' et qui habitent à casa

```
User ::where('name','like','mans%')->where('city','casa')->get()
```

Modifier les donnees du 1er user

```
>>$user=User ::find(1)
$user->name='mohamed'
```

```
$user->save()
```

Crud(create,read,update,delete) :

```
>> php artisan route:list pour voir les routes
```

Pour faire un controller qui possède les fcts de Crud

```
>>php artisan make ::controller Postcontroller --resource
```

Donc on a besoin de faire une Route

```
Route::resource('/posts',' Postcontroller '); // donc automatiquement il va générer les 7 routes
```

Si on veut générer juste la route des 2 methodes(index,show)

```
Route ::resource('/posts',' Postcontroller ')->only(['index','show']) ;
```

La method index nous permet de récupérer les donnees(posts dans notre cas)

```
public function index()
{
    dd(Person::all()); // dd :dump and die ,Person :model du table people
}
```

On peut faire aussi :

```
return view('posts.index',['allPosts'=>Person::all()]);
```

La méthode show :

```
public function show($id)
{
    dd(Person::find($id));
}
```

On peut faire :

```
return view('posts.show',['post'=>Person::find($id)]);
```

```
pour verifier si les donnees sont null dans la boucle for
on utilise
```

```
@forelse ($collection as $item)
    item!=null
```

```
@empty
    faire un msg si item==null
```

```
@endforelse
```

Il ya aussi @if @endif @else @elseif

@foreach @endforeach

Pour former les dates on utilise la library carbon : `$variableDate->diffForHumans()` // 1 day ago

- Envoyer les data via un form

First name //name :Fname
Last name //name :Lname

Dans le controller Postcontroller :
Il y a la methode create qui affiche le form

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

View :Forms/create.blade.php

```
<form method="POST" action="{{route('Posts.store')}}">
    @csrf // générer un token(laravel vérifie à chaque fois, le token est valide ou non )
    ...
</form>
```

`action="{{route('Posts.store')}}"` :il nous dirige vers la route de la méthode store et elle l'exécute.

La methode store :

Voir tous les data envoyés

Request \$request : injection de dépendance

```
public function store(Request $request)
{
    //1ere version
    dd($request->all());

    //2eme version :voir chaque élément envoyé
    $Fname=$request->input('Fname') ;
    $Lname=$request->input('Lname') ;
    dd($Fname,$Lname) ;
}
```

Pour envoyer les data à la BD : et une redirection vers la route `'Posts.index'`

```
public function store(Request $request){
    $person=new Person();
    $person->FirstName=$request->input('Fname');
    $person->LastName=$request->input('Lname');
    $person->Date_of_birth=$request->input('date');
```

```

    $person->save();
    return redirect()->route('Posts.index');
    // return redirect()->route('Posts.show',['Post'=>$person->id]);}

```

- Session flash message

Est un msg à envoyer lorsque on envoie les données avec succès

Sa durée de vie = la durée d'exécution de la requête

```

public function store(Request $request){
    . . .
    $person->save();
    $request->session()->flash('status','thank you !!! '); //nom:status
    return redirect()->route('Posts.index');
    // return redirect()->route('Posts.show',['Post'=>$person->id]);
}

```

Posts/index.blade.php :

```

@extends('layout')

@section('content')
    @if (session()->has('status'))
        <h1 style="color: green">
            {{session()->get('status')}}
        </h1>
        . . .
    @endsection

```

- Valider les champs :

1ere version :

```

public function store(Request $request){
    $request->validate(['Fname'=>'required|min:8|max:50','Lname'=>'required']);
    . . .
}

```

2eme version :

```

public function store(Request $request){
    $request->validate(['Fname'=>'bail|required|min:8|max:50','Lname'=>'required']);
    ;
    //bail : vérifier seulement la première condition
    . . .
}

```

create.blade.php :

```

@if ($errors->any())
    <ul>
        @foreach ($errors->all() as $error)
            <li>{{ $error }}</li>
        @endforeach
    </ul>

```

```
@endforeach

</ul>
@endif
```

- Associer une request à la méthode store

>> `php artisan make:request PostRequest`

Automatiquement le vs code va générer un fichier `Requests/ PostRequest.php` qui contient un classe `PostRequest` et qui hérite de la classe `FormRequest` donc la méthode devient :

```
public function store(PostRequest $request){. . .}
```

et on va supprimer :

```
$request->validate(['Fname'=>'bail|required|min:8|max:50','Lname'=>'required'])
```

O va mettre `['Fname'=>'bail|required|min:8|max:50','Lname'=>'required']` dans la methode rules .

Donc la classe `PostRequest` devient :

```
class PostRequest extends FormRequest
{
    * @return bool
    public function authorize()
    {
        return true;// remplacer false par true
    }
    * @return array
    public function rules()
    {
        return ['Fname'=>'required|min:8|max:50','Lname'=>'required'];
    }
}
```

- Si on veut garder les infos après l'actualisation de la page :

```
<input type="text" id="FN" class="form-control" placeholder="First name" name="Fname" value="{{old('Fname')}}">
```

- Modifier les data dans la DB

Si on veut autoriser tout sauf la méthode `DESTROY` :

```
Route::resource('Posts',PostsController)->except('destroy');
```

- Créer un Forms + édit
 - View :`edit.blade.php`

```
@extends('layout')

@section('content')
```

```

<form method="POST" action="{{ route('Posts.update', ['Post'=>$person->id]) }}" >
    @csrf //token
    @method('PUT') // appliquer la methode PUT
    <label for="ID">ID</label>
    <input type="text" id="ID" name="id" value="{{ $person->id }}" >
    ... //les autres inputs
    <button type="submit" class="btn btn-primary">Update</button>
</form>
@endsection

```

▪ Controller

```

public function edit($id)
{
    $person=Person::findOrFail($id);
    //Fail : afficher 404 au cas d'erreur
    return view('Forms.edit', ['person'=>$person]);
}

public function update(PostRequest $request, $id){
    $person=Person::findOrFail($id);
    $person->id=$request->input('id');
    $person->FirstName=$request->input('Fname');
    $person->LastName=$request->input('Lname');
    $person->Date_of_birth=$request->input('date');
    $person->save();
    return redirect()->route('Posts.index');
    //
}

```

- Si on veut inclure un fichier dans extends

```

• @extends('layout')
•
• @section('content')
•     . . .
•     @include('name') // inclure le fichier name.blade.php
•     . . .
•
• @endsection

```

View : name.blade.php

```

. . .
<input type="text" name="id" value="{{ old('id', $person->id) }}" >
. . .

```

Si la variable `$person` n'existe pas dans d'autres fichiers → un error
Donc on fait la syntaxe suivante :

```

<input type="text" name="id" value="{{ old('id', $person->id ?? null) }}" >

```

- Delete data via tinker
 >>\$person=Person::find(11)
 >>\$person->delete()

>>Person::destroy(11)

>>Person::destroy([1,2,3])

- Supprimer un élément via une view

Activer toutes les routes

```
Route::resource('Posts','App\Http\Controllers\PostsController');
```

```
public function destroy(Request $request,$id){

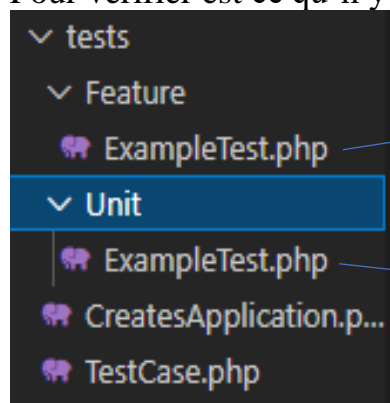
    // Person::destroy($id);

    $person=Person::findOrFail($id);
    $person->delete();
    $request->session()->flash('status','The person was deleted');
    return redirect()->route('Posts.index');
}
```

```
<td>
//dans le champs action
<a href="{{route('Posts.edit',['Post'=>$person->id])}}" >edit</a>
<form action="{{route('Posts.destroy',['Post'=>$person->id])}}" method="POST">
    @csrf
    @method('DELETE')
    <button type="submit" class="btn btn-danger">delete</button>
</form>
</td>
```

- Testing

Pour vérifier est ce qu'il y a des erreurs dans des fonctionnalités



Pour les tests compliqués

Pour les tests simples

On utilise souvent Feature

On teste la route '/' on peut tester d'autres routes (`$this->get('/')`;)

`$response->assertStatus(200)`; s'il return 200 → la page fonctionne bien

Sinon return 404,... → error

```
class ExampleTest extends TestCase
{
    /**
     * A basic test example.
     *
     * @return void
     */
    public function testBasicTest()
    {
        $response = $this->get('/');

        $response->assertStatus(200);
    }
}
```

Pour Unit (vérification des données) s'il return true → pas d'error

Sinon (false) → error

Pour faire le test on utilise le cmd :


```
C:\xampp\htdocs\coursLaravel>vendor\bin\phpunit
PHPUnit 9.5.4 by Sebastian Bergmann and contributors.

..
Time: 00:00.412, Memory: 20.00 MB
OK (2 tests, 2 assertions)
```

Pour se connecter à la BD → fichier **.env**

- Faire un teste sur DB(fausse DB) qui existe dans la mémoire(RAM)u PC

⇒

 phpunit.xml

```
<server name="APP_ENV" value="testing"/>
. . .
<server name="DB_CONNECTION" value="sqlite_testing"/>//on ajoute cette ligne
. . .
<server name="TELESCOPE_ENABLED" value="false"/>
```

⇒ Config/database.php

On ajoute ce code :

```
'sqlite_testing' => [
    'driver' => 'sqlite',
    'database' => ':memory:', //créer une db dans la RAM
],
```

⇒ Pour appliquer ces changements :il faut relancer le serveur ou vider le cache

Vider cache → cmd :>> php artisan config:clear

- Faire votre propre teste

Tester des routes et vérifier est ce que les views (→routes) contiennent des textes (il y a d'autres assertions, voir documentation)

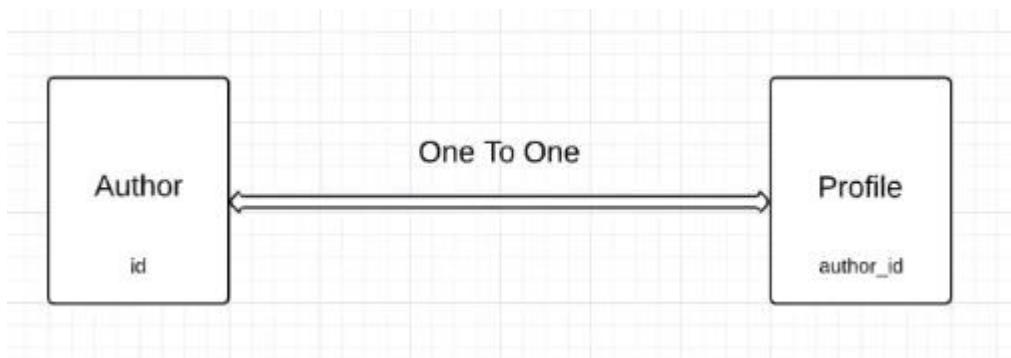
Cmd :>> php artisan make:test Peopletest

Nom du classe : People+test , le premier char doit être en majuscule

```
class Peopletest extends TestCase
{
    public function test_example()
    {
        $response = $this->get('/Posts');
        $response->assertSeeText('ayoub');//la page contient 'ayoub' ?
    }
}
```

Cmd >> vendor/bin/phpunit → resultat

- Testing DB (voir brightcoding)
- **One To One relation with migration**



id : author_PK

author_id : author_FK → chaque author a un unique profile

```

C:\laragon\www\laravel-course
λ php artisan make:model Author --migration
Model created successfully.
Created Migration: 2019_09_19_083538_create_authors_table

C:\laragon\www\laravel-course
λ php artisan make:model Profile --migration
Model created successfully.
Created Migration: 2019_09_19_083839_create_profiles_table
  
```

```

public function up(){
    Schema::create('authors', function (Blueprint $table) {
        $table->bigIncrements('id');
        $table->timestamps();
    });
}
  
```

```

public function up(){
    Schema::create('profiles', function (Blueprint $table) {
        $table->bigIncrements('id');
        $table->unsignedBigInteger('author_id')->unique();
        $table->foreign('author_id')->references('id')->on('authors');
        $table->timestamps();
    });
}
  
```

Cmd `>> php artisan migrate` → création des tables dans la BD

On a besoin de faire une liaison entre le model author et profile (chaque author a un unique profile)

```
class author extends Model{
    use HasFactory;
    public function Profile(){
        return $this->hasOne('App\profile');
    }
}
```

```
class profile extends Model{
    use HasFactory;
    public function Author(){
        return $this->belongsTo('App\author');
    }
}
```

- **One To One assignment relationship (faire la liaison)**

```
• public function show_Relations(){
•     return view('One_OneAndOne_many'); ➔ button :create author&&Profile
• }
```

```
public function create_author_profile(Request $request){
    $author=new author();
    $author->save();

    $profile=new profile();
    $profile->Author()->associate($author)->save();
    $request->session()->flash('status','the author was created');
    return redirect()->route('relation');
}
```

- One To One querying relationship (récupération des données)

Chaque auteur avec son profile

```
>>> $authors = Author::with('profile')->get()
=> Illuminate\Database\Eloquent\Collection {#3071
  all: [
    App\Author {#3079
      id: 1,
      created_at: "2019-09-19 09:34:53",
      updated_at: "2019-09-19 09:34:53",
      profile: App\Profile {#3086
        id: 1,
        author_id: 1,
        created_at: "2019-09-19 09:35:36",
        updated_at: "2019-09-19 09:35:36",
      },
    },
    App\Author {#3080
      id: 2,
      created_at: "2019-09-19 09:48:03",
      updated_at: "2019-09-19 09:48:03",
      profile: App\Profile {#3088
        id: 2,
        author_id: 2,
        created_at: "2019-09-19 09:49:17",
        updated_at: "2019-09-19 09:49:17",
      },
    },
  ],
}
```

Author qui a un id=1 avec son profile

```
>>> $author = Author::with('profile')->whereId(1)->get()
=> Illuminate\Database\Eloquent\Collection {#3082
  all: [
    App\Author {#3063
      id: 1,
      created_at: "2019-09-19 09:34:53",
      updated_at: "2019-09-19 09:34:53",
      profile: App\Profile {#3099
        id: 1,
        author_id: 1,
        created_at: "2019-09-19 09:35:36",
        updated_at: "2019-09-19 09:35:36",
      },
    },
  ],
}

>>> $author = Author::with('profile')->whereId(1)->first()
=> App\Author {#3093
  id: 1,
  created_at: "2019-09-19 09:34:53",
  updated_at: "2019-09-19 09:34:53",
  profile: App\Profile {#3105
    id: 1,
    author_id: 1,
    created_at: "2019-09-19 09:35:36",
    updated_at: "2019-09-19 09:35:36",
  },
}
```

Methode 1 :

Affiche
plusieurs

Methode 2 :

Affiche 1 seule
objet

Chaque profile avec son author

```
>>> $profiles = Profile::with('author')->get()
=> Illuminate\Database\Eloquent\Collection {#3103
  all: [
    App\Profile {#3109
      id: 1,
      author_id: 1,
      created_at: "2019-09-19 09:35:36",
      updated_at: "2019-09-19 09:35:36",
      author: App\Author {#3116
        id: 1,
        created_at: "2019-09-19 09:34:53",
        updated_at: "2019-09-19 09:34:53",
      },
    },
    App\Profile {#3110
      id: 2,
      author_id: 2,
      created_at: "2019-09-19 09:49:17",
      updated_at: "2019-09-19 09:49:17",
      author: App\Author {#3118
        id: 2,
        created_at: "2019-09-19 09:48:03",
        updated_at: "2019-09-19 09:48:03",
      },
    },
  ],
}
```

Profile qui a un id=2 avec son profile

```
>>> $profiles = Profile::with('author')->first()
=> App\Profile {#3101
  id: 1,
  author_id: 1,
  created_at: "2019-09-19 09:35:36",
  updated_at: "2019-09-19 09:35:36",
  author: App\Author {#3122
    id: 1,
    created_at: "2019-09-19 09:34:53",
    updated_at: "2019-09-19 09:34:53",
  },
}

>>> $profiles = Profile::with('author')->whereKey(2)->first()
=> App\Profile {#3035
  id: 2,
  author_id: 2,
  created_at: "2019-09-19 09:49:17",
  updated_at: "2019-09-19 09:49:17",
  author: App\Author {#3125
    id: 2,
    created_at: "2019-09-19 09:48:03",
    updated_at: "2019-09-19 09:48:03",
  },
}
```

- Email vérification /login /logout

- 1) Installer laravel/ui
>> composer require laravel/ui
- 2) Installer le système d'auth
>> php artisan ui vue --auth
- 3) Configuration .env

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=5dc4f54d1c21b0
MAIL_PASSWORD=b21137dba66545
MAIL_ENCRYPTION=null
MAIL_FROM_ADDRESS=null
MAIL_FROM_NAME="${APP_NAME}"
```

- 4) Ajouter la ligne « implements MustVerifyEmail sur le model User »

```
class User extends Authenticatable implements MustVerifyEmail
{
```

```
protected $fillable = [
    'First_Name',
    'Last_Name',
    'phone',
    'Date_Of_birth',
    'email',
    'password',
];
```

- 5) Dire qu'avant exécuter les autres routes, il faut que le mail soit vérifié.

```
Auth::routes(['verify'=>true]);
```

- 6) Check HomeController

```
class HomeController extends Controller
{
    /**
     * Create a new controller instance.
     *
     * @return void
     */
    public function __construct()
    {
        $this->middleware('auth');//user must be authenticated
    }
}
```

- 7) RegisterController (validation,store)

```
protected function create(array $data)
{
    return User::create([
        'First_Name' => $data['First_Name'],
        'Last_Name' => $data['Last_Name'],
        'Date_Of_birth' => $data['Date_Of_birth'],
        'email' => $data['email'],
        'phone' => $data['phone'],
    ]);
}
```

```

        'password' => Hash::make($data['password']),
    ]);
}

```

8) LoginController

Ajouter la methode logout et aussi les namespaces de (request,auth)

```

public function logout(Request $request)
{
    Auth::logout();//on doit ajouter le namespace de Auth
    $request->session()->invalidate();//supprimer la session
    $request->session()->regenerateToken();//regenerer le token
    return $this->loggedOut($request) ?: redirect('/login');
}

```

9) Les routes

Avant d'aller à la home , il faut que le mail soit vérifié(middleware('verified'))

```

Auth::routes(['verify'=>true]);
Route::get('/home', [App\Http\Controllers\HomeController::class, 'index']
->name('home')->middleware('verified'));

```

10) Les views

Login.blade.php

```

<form method="POST" action="{{ route('login') }}">
    @csrf
    . . .
    <div>
        <input type="checkbox" name="remember" id="remember" {{ old('remember')
? 'checked' : '' }}>
        <label for="remember">Se souvenir de moi</label>
    </div>
    <button type="submit">Se connecter</button>
    <div style="text-align: center">
        @if (Route::has('password.request'))
            <a href="{{ route('password.request') }}">
                {{ __('Mot de passe oublié ?') }}
            </a>.
        @endif
        <a href="{{ route('register') }}">S'inscrire </a>
    </div>

</form>

```

register.blade.php

```

<form method="POST" action="{{ route('register') }}">

```



```
@csrf
</form>
```

home.blade.php

```
@extends('layout')
@section('content')
    <h1 style="color:green;text-align:center">Welcome</h1>
    <form action="{{route('logout')}}" method="POST">
        @csrf
        <button type="submit" class="btn btn-primary">Se deconnecter</button>
    </form>
@endsection
```

11) How to prevent going back after logout in laravel

```
>> php artisan make:middleware PreventBackHistory
```

- **Middleware Configuration**

Open up PreventBackHistory.php file in app/Http/Middleware folder and replace codes with the following codes below: **app/Http/Middleware/PreventBackHistory.php**

```
<?php
namespace App\Http\Middleware;
use Closure;
class PreventBackHistory {
    /**
     * Handle an incoming request.
     * @param \Illuminate\Http\Request $request
     * @param \Closure $next
     * @return mixed
     */
    public function handle($request, Closure $next) {
        $response = $next($request);
        return $response->header('Cache-Control','nocache, no-store, max-age=0, must-revalidate')
            ->header('Pragma','no-cache')
            ->header('Expires','Sat, 26 Jul 1997 05:00:00 GMT');
    }
}
```

- **Register Middleware**
app/Http/Kernel.php

```
<?php
namespace App\Http;
use Illuminate\Foundation\Http\Kernel as HttpKernel;
class Kernel extends HttpKernel {
    /**
     * The application's route middleware.
     * These middleware may be assigned to groups or used individually.
     */
}
```

```

* @var array
*/
protected $routeMiddleware = [
    'prevent-back-history' => \App\Http\Middleware\PreventBackHistory::class,
];
}

```

- **Add Middleware in Route**

routes/web.php

```

Route::group(['middleware' => 'prevent-back-history'],function(){
    Auth::routes();
    Route::get('/home', 'HomeController@index');
});

```