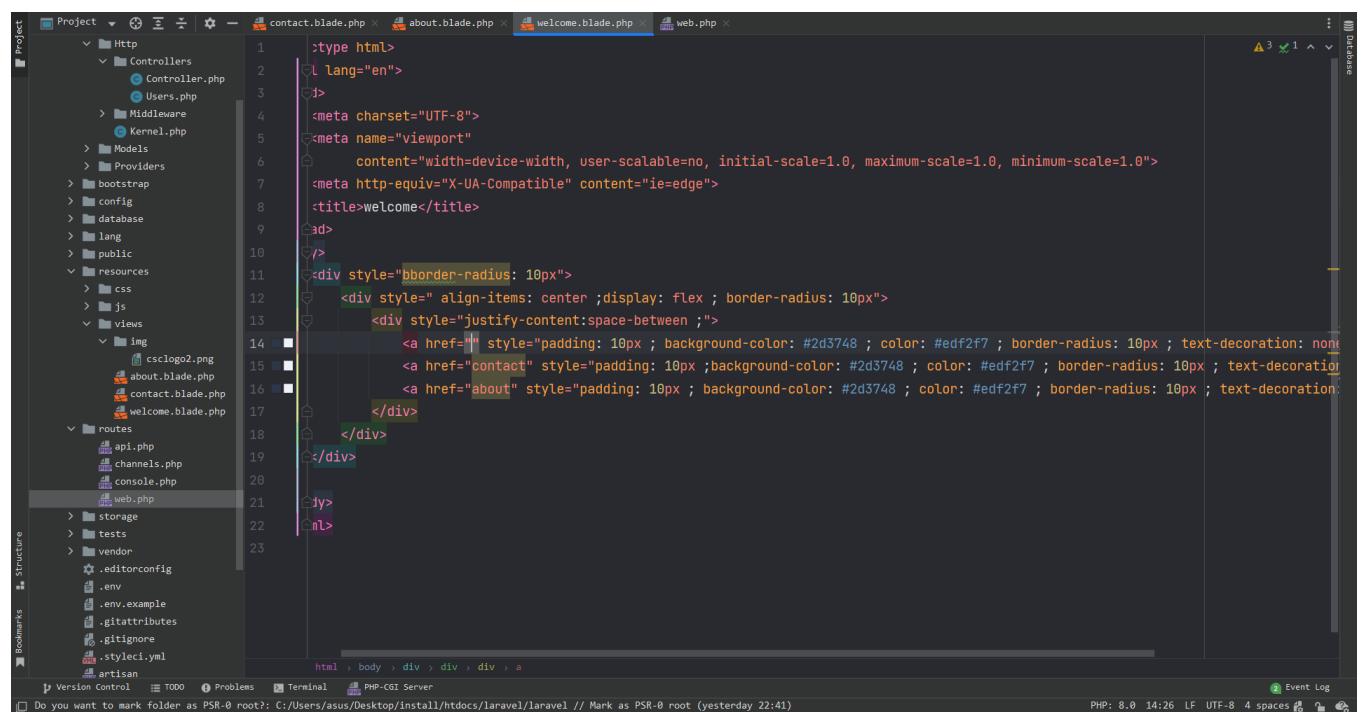


# Laravel 9

Achraf KHABAR

vendredi 25 février 2022  
00:20

## Routing :



The screenshot shows a code editor with the following details:

- Project Structure:** Shows the Laravel directory structure including Http, Middleware, Providers, config, database, lang, public, resources, routes, storage, tests, vendor, .env, .env.example, .gitattributes, .gitignore, .styleci.yml, and artisan.
- File Content:** A partial view file named `welcome.blade.php` which contains the following HTML and Blade code:

```
<type html>
<lang="en">
</>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
<meta http-equiv="X-UA-Compatible" content="ie=edge">
<title>welcome</title>
<div style="border-radius: 10px">
    <div style=" align-items: center ;display: flex ; border-radius: 10px">
        <div style="justify-content:space-between ;">
            <a href="" style="padding: 10px ;background-color: #2d3748 ; color: #edf2f7 ; border-radius: 10px ; text-decoration: none">
                Contact
            </a>
            <a href="contact" style="padding: 10px ;background-color: #2d3748 ; color: #edf2f7 ; border-radius: 10px ; text-decoration: none">
                About
            </a>
            <a href="about" style="padding: 10px ;background-color: #2d3748 ; color: #edf2f7 ; border-radius: 10px ; text-decoration: none">
                Home
            </a>
        </div>
    </div>
</div>
```
- Terminal:** Shows the command `Do you want to mark folder as PSR-0 root? C:/Users/asus/Desktop/install/htdocs/laravel/laravel // Mark as PSR-0 root (yesterday 22:41)`.
- Bottom Bar:** Shows PHP version 8.0, memory usage 14:26, and other system information.



The screenshot shows the `routes/web.php` file with the following content:

```
Route::view( uri: "/" , view: "welcome" ) ;
Route::view( uri: "Activities" , view: "Activities" ); // the links was declared
Route::view( uri: "about" , view: "about" ) ;
```

I can return such things not only views :

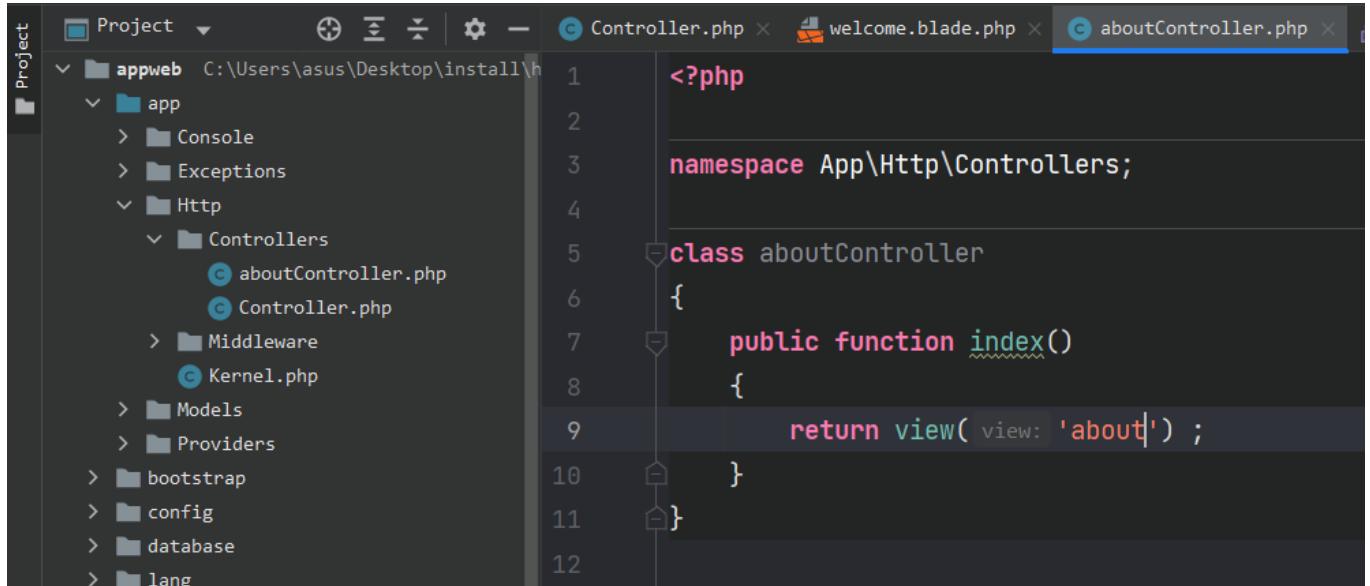


The screenshot shows a controller method in `postsController.php` with the following code:

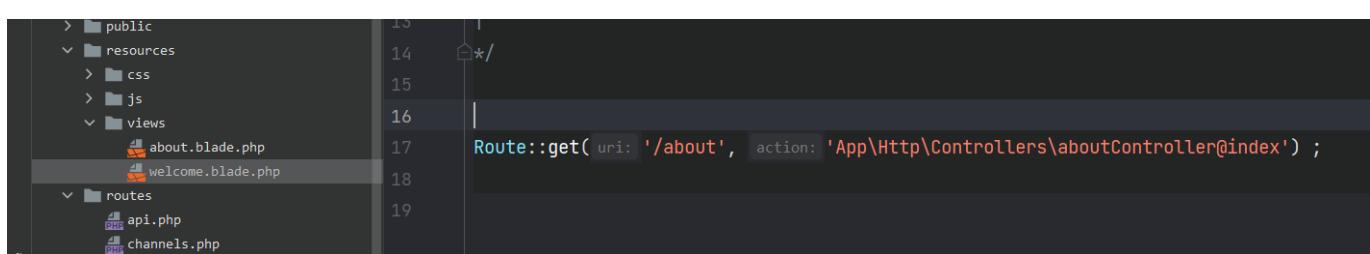
```
Route::get( uri: '/postes' , function (){
    return response()->json([
        'title' => 'mon super title' ,
        'description' => 'ma super description '
    ]);
});
```

```
{"title":"mon super title","description":"ma super description "}
```

## Controllers :



```
<?php  
  
namespace App\Http\Controllers;  
  
class aboutController  
{  
    public function index()  
    {  
        return view('about');  
    }  
}
```



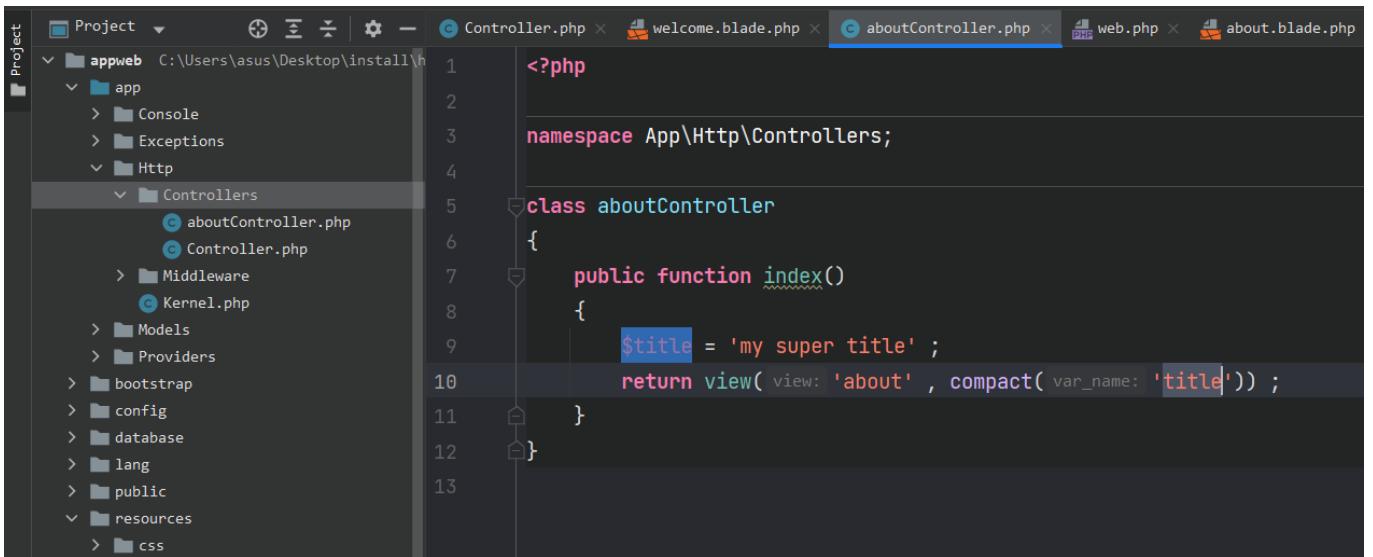
```
Route::get('/about', [aboutController@index]);
```

An other way :

```
use App\Http\Controllers\aboutController;
use Illuminate\Support\Facades\Route;

/*
|--------------------------------------------------------------------------
| 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('uri: /about', [aboutController::class , 'index']);
```

## How to use a variable in laravel :



The screenshot shows a code editor with the following details:

- Project Structure:** The left sidebar shows the project structure under "appweb". It includes "app" (Console, Exceptions, Http), "Http" (Controllers, Middleware, Kernel.php), "Models", "Providers", "bootstrap", "config", "database", "lang", "public", and "resources" (css).
- Open Files:** The tabs at the top show "Controller.php", "welcome.blade.php", "aboutController.php" (which is the active tab), "web.php", and "about.blade.php".
- Code Content:** The "aboutController.php" file contains the following code:

```
<?php

namespace App\Http\Controllers;

class aboutController
{
    public function index()
    {
        $title = 'my super title';
        return view( view: 'about' , compact( var_name: 'title')) ;
    }
}
```

The screenshot shows a file structure on the left and a blade template file on the right.

**File Structure:**

```

structure
    Controllers
        aboutController.php
        Controller.php
        Middleware
            Kernel.php
    Models
    Providers
    bootstrap
    config
    database
    lang
    public
    resources
        css
        js
    views
        about.blade.php
        welcome.blade.php

```

**blade.php Content:**

```

5   <meta name="viewport"
6       content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
7   <meta http-equiv="X-UA-Compatible" content="ie=edge">
8   <title>about</title>
9   </head>
10  <body>
11      <div style="text-align: center;">
12          <h1>
13              print la balle
14          </h1>
15          <h1>
16              {{ $title }}
17          </h1>
18      </div>
19
20  </body>
21  </html>

```

An other way :

The screenshot shows a file structure on the left and two files: a controller class and a blade template.

**File Structure:**

```

structure
    Console
    Exceptions
    Http
        Controllers
            aboutController.php
            Controller.php
        Middleware
            Kernel.php
    Models
    Providers
    bootstrap
    config
    database
    lang
    public

```

**aboutController.php Content:**

```

namespace App\Http\Controllers;

class aboutController
{
    public function index()
    {
        $title = 'my super title';
        return view('about')->with('title', $title);
    }
}

```

**blade.php Content:**

```

5   <meta name="viewport"
6       content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
7   <meta http-equiv="X-UA-Compatible" content="ie=edge">
8   <title>about</title>
9   </head>
10  <body>
11      <div style="text-align: center;">
12          <h1>
13              print la balle
14          </h1>
15          <h1>
16              {{ $title }}
17          </h1>
18      </div>
19
20  </body>
21  </html>

```

## How to pass more than variable :

```

2
3     namespace App\Http\Controllers;
4
5     class aboutController
6     {
7         public function index()
8         {
9             $title = 'my super title';
10            $title2 = 'my super second title';
11
12            return view('about', compact('title', ...$title2));
13        }
14    }

```

```

6           content="width=device-width, user-scalable=no, initial-scale=1, maximum-scale=1">
7           <meta http-equiv="X-UA-Compatible" content="ie=edge">
8           <title>about</title>
9           </head>
10          <body>
11              <div style="text-align: center;">
12                  <h1>
13                      print la balle
14                  </h1>
15                  <h1>
16                      {{ $title }} <br>
17                      {{ $title2 }} <br>
18                  </h1>
19              </div>
20
21          </body>
22      </html>

```

Or we can do an array :

```

9           $title = 'my super title';
10          $title2 = 'my super second title';
11
12          return view('about', [
13              'title' => $title,
14              'title2' => $title2
15          ]);
16      }
17  }
18 }

```

## How to pass them within array :

The screenshot shows a code editor with two panes. The left pane displays the project structure:

```

> console
> Exceptions
> Http
  > Controllers
    < aboutController.php
    < Controller.php
    > Middleware
      < Kernel.php
    > Models
    > Providers
  > bootstrap
  > config
  > database
  > lang
  > public
  > resources

```

The right pane shows the `aboutController.php` file:

```

3 namespace App\Http\Controllers;
4
5 class aboutController
6 {
7     public function index()
8     {
9         $titles = [ 'my super title' , 'my super second title' ] ;
10
11         return view( view: 'about' , compact( var_name: '|titles|'));
12     }
13 }
14

```

Below this, another code editor pane shows the `about.blade.php` template:

```

11 <div style="text-align: center;">
12   <h1>
13     print la balle
14   </h1>
15   <h1>
16     @foreach( $titles as $title)
17       <h6>
18         {{ $title }}
19       </h6>
20     @endforeach
21   </h1>
22 </div>

```

## Blade and views :

The screenshot shows a code editor with two panes. The left pane displays the project structure:

```

> storage
> config
> database
> lang
> public
> resources
  > css
  > js
  > views
    < about.blade.php
    < welcome.blade.php
  > routes
    < api.php
    < channels.php
    < console.php
    < web.php
> storage

```

The right pane shows the `routes/web.php` file:

```

18
19 Route::get( uri: '/about' , [aboutController::class , 'index']) ;
20 Route::get( uri: '/about/{id}' , [aboutController::class , 'show']) ;

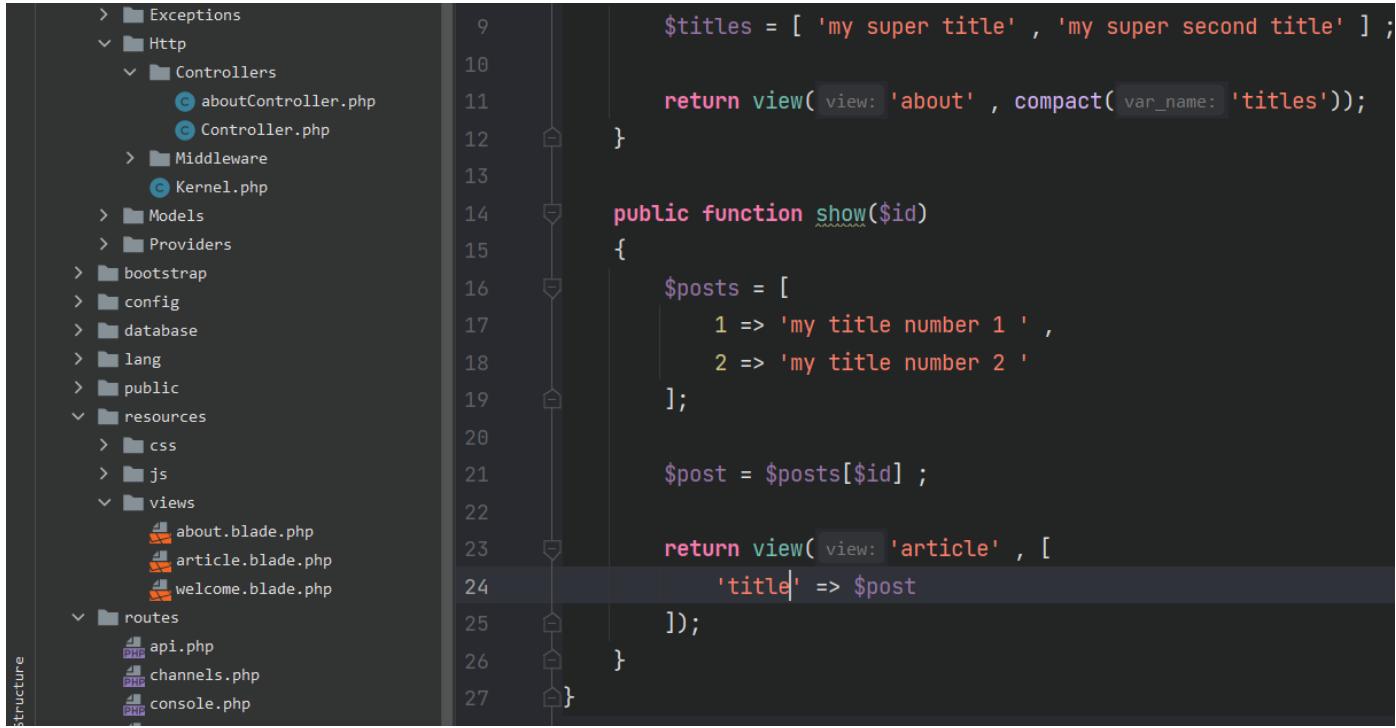
```

Below this, another code editor pane shows the `about.blade.php` template:

```

Route::get( uri: '/about' , [aboutController::class , 'index']) ;
Route::get( uri: '/about/posts/{id}' , [aboutController::class , 'show']) ;

```



The screenshot shows the PHPStorm IDE interface. On the left, the 'Structure' tool window displays the project's directory structure:

```

> Exceptions
<-- Http
    <-- Controllers
        aboutController.php
        Controller.php
    <-- Middleware
        Kernel.php
<-- Models
<-- Providers
> bootstrap
> config
> database
> lang
> public
<-- resources
    <-- css
    <-- js
    <-- views
        about.blade.php
        article.blade.php
        welcome.blade.php
<-- routes
    api.php
    channels.php
    console.php

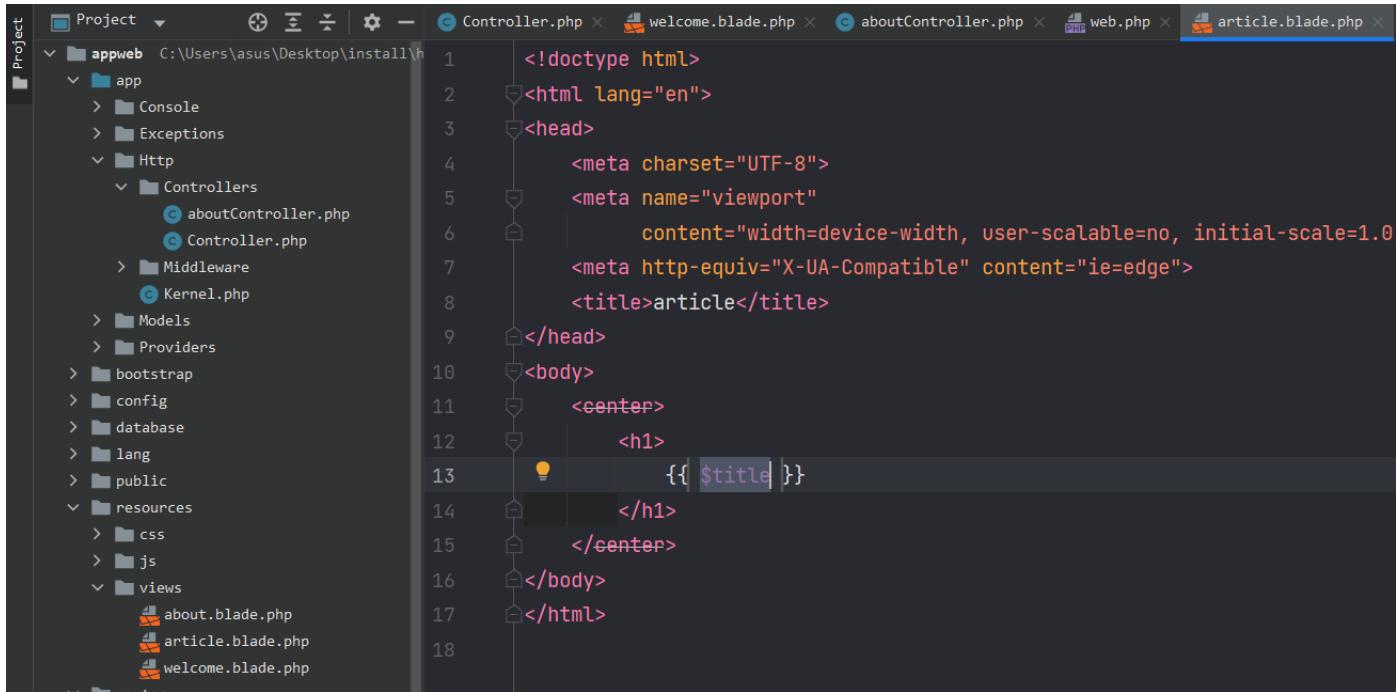
```

The main code editor window contains the following PHP code from the `Controller.php` file:

```

9     $titles = [ 'my super title' , 'my super second title' ];
10
11    return view( view: 'about' , compact(var_name: 'titles'));
12
13
14    public function show($id)
15    {
16        $posts = [
17            1 => 'my title number 1 ' ,
18            2 => 'my title number 2 '
19        ];
20
21        $post = $posts[$id] ;
22
23        return view( view: 'article' , [
24            'title' => $post
25        ]);
26    }
27

```



The screenshot shows the PHPStorm IDE interface. On the left, the 'Project' tool window displays the project's directory structure:

```

appweb C:\Users\asus\Desktop\install\
    <-- app
        <-- Console
        <-- Exceptions
        <-- Http
            <-- Controllers
                aboutController.php
                Controller.php
            <-- Middleware
                Kernel.php
        <-- Models
        <-- Providers
        <-- bootstrap
        <-- config
        <-- database
        <-- lang
        <-- public
        <-- resources
            <-- css
            <-- js
            <-- views
                about.blade.php
                article.blade.php
                welcome.blade.php

```

The main code editor window contains the following Blade template code from the `article.blade.php` file:

```

1     <!doctype html>
2     <html lang="en">
3         <head>
4             <meta charset="UTF-8">
5             <meta name="viewport"
6                 content="width=device-width, user-scalable=no, initial-scale=1.0
7             <meta http-equiv="X-UA-Compatible" content="ie=edge">
8             <title>article</title>
9         </head>
10        <body>
11            <center>
12                <h1>
13                    {{ $title }}
14                </h1>
15            </center>
16        </body>
17    </html>
18

```

Add condition :

The screenshot shows a file structure on the left and a code editor on the right. The file structure includes:

- Middleware**: Contains `Kernel.php`.
- Models**
- Providers**
- bootstrap**
- config**
- database**
- lang**
- public**
- resources** (selected):
  - css**
  - js**
  - views** (selected):
    - `about.blade.php`
    - `article.blade.php`
    - `welcome.blade.php`
- routes** (selected):
  - `api.php`
  - `channels.php`
  - `console.php`
  - `web.php`

The code editor displays a PHP class definition:

```

13
14     public function show($id)
15     {
16         $posts = [
17             1 => 'my title number 1' ,
18             2 => 'my title number 2'
19         ];
20
21         $post = $posts[$id] ?? 'no title exist now' ;
22
23         return view( view: 'article' , [
24             'title' => $post
25         ]);
26     }
27 }
28

```

Aussi :

The screenshot shows a file structure on the left and a code editor on the right. The file structure includes:

- lang**
- public**
- resources** (selected):
  - css**
  - js**
  - views** (selected):
    - `about.blade.php`
    - `article.blade.php`
    - `welcome.blade.php`
- routes** (selected):
  - `api.php`
  - `channels.php`
  - `console.php`
  - `web.php`
- storage**

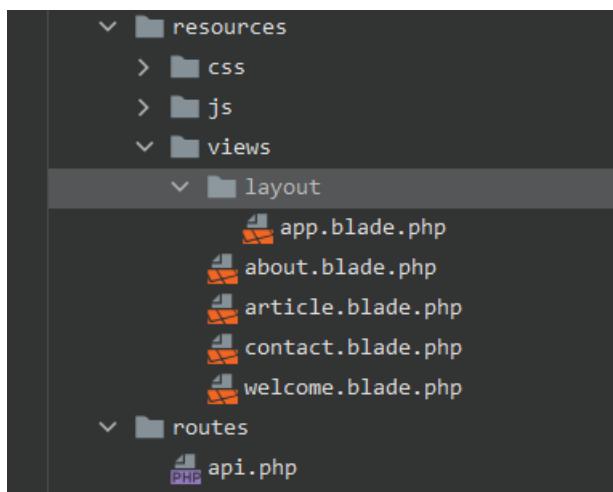
The code editor displays a PHP code snippet:

```

12 | routes are loaded by the RouteServiceProvider within a group which
13 | contains the "web" middleware group. Now create something great!
14 |
15 | */
16
17 Route::view( uri: '/welcome' , view: 'welcome' );
18
19 Route::get( uri: '/about' , [aboutController::class , 'index']) ;
20 Route::get( uri: '/about/{id}' , [aboutController::class , 'show'])->whereNumber( parameters: 'id' ) ;
21 Route::get( uri: '/about/contact' , [aboutController::class , 'contact']) ;
22
23

```

## Advantage of .blade (how to not repeate the coman code between views) :



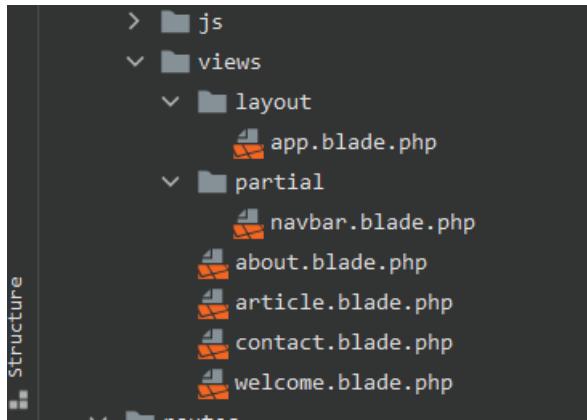
A screenshot of a code editor showing a layout blade file named 'app.blade.php'. The code defines a basic HTML structure with a head section containing meta tags for charset, viewport, and title, and a body section with a yield placeholder for content.

```
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport"
6       content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
7     <meta http-equiv="X-UA-Compatible" content="ie=edge">
8     <title>AKC</title>
9   </head>
10  <body>
11    @yield('content')
12  </body>
13 </html>
```

A screenshot of a code editor showing a content blade file named 'about.blade.php'. It extends the 'layout.app' layout and defines a section for content. The content includes a large title 'print la balle', a foreach loop to iterate over titles, and an endsection.

```
1 @extends('layout.app')
2
3 @section('content')
4   <div class="premiere-div" style="...>
5     <h1 class="grand-titre">
6       print la balle
7     </h1>
8     <h1>
9       @foreach( $titles as $title)
10      <h6>
11        {{ $title }}
12      </h6>
13      @endforeach
14    </h1>
15  </div>
16 @endsection
```

**Include in blade :**



```
1 <ul>
2   <li><a href="#">Home</a></li>
3   <li><a href="#">About</a></li>
4   <li><a href="#">Contact</a></li>
5
6 </ul>
```

```
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport"
6       content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
7     <meta http-equiv="X-UA-Compatible" content="ie=edge">
8     <title>AKC</title>
9   </head>
10  <body>
11    @include('partial.navbar')
12    @yield('content')
13  </body>
14  </html>
```

The screenshot shows the project structure on the left with the 'appweb' folder expanded. The 'views' directory is selected. On the right, the code editor displays the content of 'app.blade.php'. It includes an HTML doctype, an HTML tag with the 'lang' attribute set to 'en', a head section with meta tags for charset, viewport, and X-UA-Compatible, a title tag with the value 'AKC', and a body section. Inside the body, there is an '@include' directive for 'partial.navbar' and an '@yield' directive for 'content'.

**How to name routes :**

```

16
17 Route::view( uri: '/welcome', view: 'welcome')->name( name: 'welcome') ;
18
19 Route::get( uri: '/about', [aboutController::class , 'index'])->name( name: 'about') ;
20 Route::get( uri: '/about/{id}', [aboutController::class , 'show'])->whereNumber( parameters: 'id')->name( name: 'about-id') ;
21 Route::get( uri: '/about/contact', [aboutController::class , 'contact'])->name( name: 'contact') ;
22

```

```

1 <ul>
2   <li><a href="{{ route('welcome') }}>Home</a></li>
3   <li><a href="{{ route('about') }}>About</a></li>
4   <li><a href="{{ route('contact') }}>Contact</a></li>
5
6 </ul>
7

```

## Base de données avec laravel :

```

4
5   return [
6     /*
7      |--------------------------------------------------------------------------
8      | Default Database Connection Name
9      |--------------------------------------------------------------------------
10     |
11     |
12     | Here you may specify which of the database connections below you wish
13     | to use as your default connection for all database work. Of course
14     | you may use many connections at once using the Database library.
15     |
16   */
17
18   'default' => env( key: 'DB_CONNECTION', default: 'mysql'),
19

```

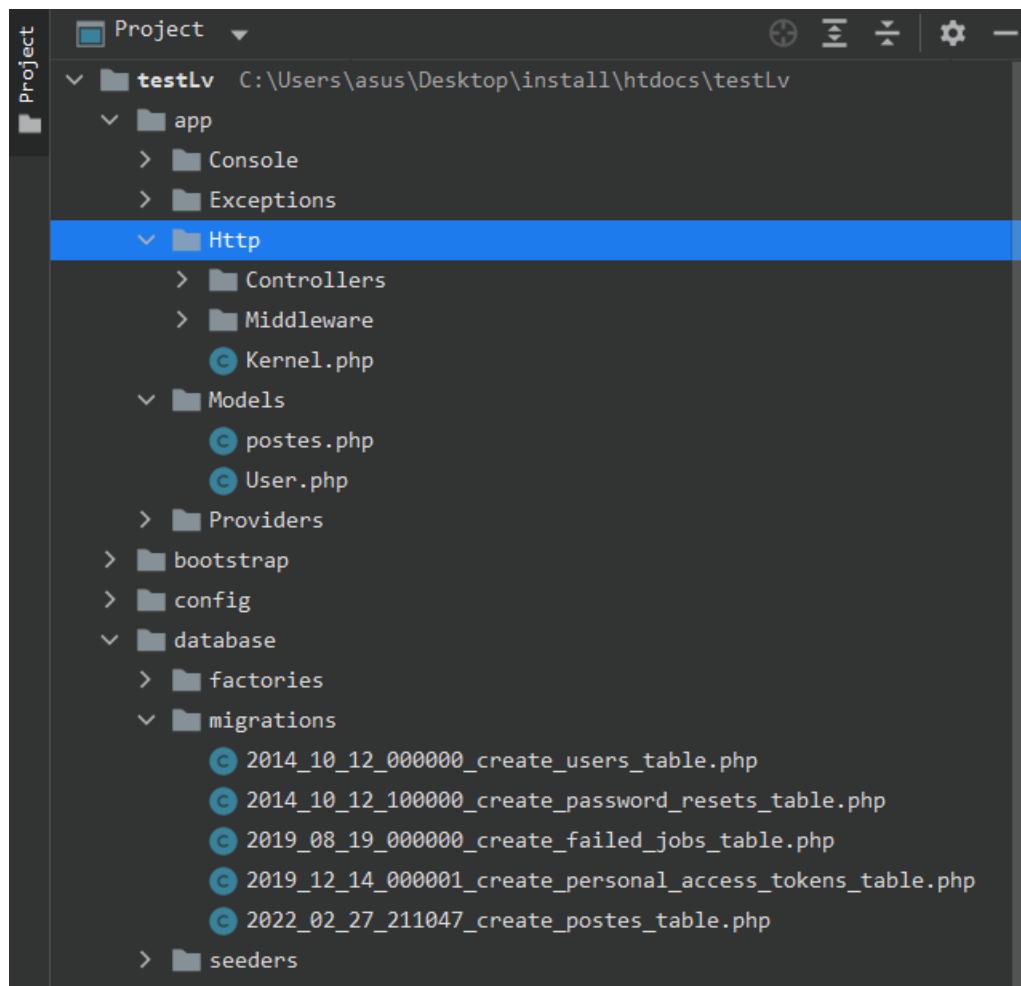
```

11 DB_CONNECTION=mysql
12 DB_HOST=127.0.0.1
13 DB_PORT=3306
14 DB_DATABASE=laravel
15 DB_USERNAME=root
16 DB_PASSWORD=
17

```

## How to create a model and his migration ( migration in order to create tables ) :

```
PS C:\Users\asus\Desktop\install\htdocs\testLv> php artisan make:model postes -m
```



## Create a database in laravel 8 :

The screenshot shows a code editor displaying a PHP migration file. The file is located in the 'migrations' directory under the 'database' namespace. The code defines a 'up()' function that creates a table named 'postes' with specific columns and constraints.

```
public function up()
{
    Schema::create('postes', function (Blueprint $table) {
        $table->id();
        $table->string('title');
        $table->mediumText('content_post');
        $table->timestamps();
    });
}
```

```
Structure
Bookmarks
Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6

PS C:\Users\asus\Desktop\install\htdocs\testLv> php artisan make:model postes -m
Model created successfully.
Created Migration: 2022_02_27_211047_create_postes_table
PS C:\Users\asus\Desktop\install\htdocs\testLv> php artisan migrate
```

```
2014_10_12_000000_create
Terminal: Local × + ▾
PS C:\Users\asus\Desktop\install\htdocs\testLv> php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (84.19ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (70.25ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (250.67ms)
Migrating: 2019_12_14_000001_create_personal_access_tokens_table
Migrated: 2019_12_14_000001_create_personal_access_tokens_table (134.35ms)
Migrating: 2022_02_27_211047_create_postes_table
Migrated: 2022_02_27_211047_create_postes_table (30.17ms)
PS C:\Users\asus\Desktop\install\htdocs\testLv>
```

**Eloquent : ( fetch data from the data base )**

The screenshot shows a PHP IDE interface with a project named 'testLv'. The left sidebar displays the project structure:

- app
  - Console
  - Exceptions
  - Http
  - Models
    - games.php
    - postes.php
    - User.php
  - Providers
  - bootstrap
  - config
  - database
  - lang
  - node\_modules library root
  - public

The right pane shows the code for 'games.php':

```
<?php  
namespace App\Models;  
  
use Illuminate\Database\Eloquent\Factories\HasFactory;  
use Illuminate\Database\Eloquent\Model;  
  
class games extends Model  
{  
    use HasFactory;  
}
```

The screenshot shows a PHP IDE interface with a project structure:

- public
- resources
  - css
  - js
  - views
    - layout
    - partial
      - navbar.blade.php
      - about.blade.php
      - contact.blade.php
      - games.blade.php
      - home.blade.php
  - routes
    - api.php
    - channels.php
    - console.php
    - web.php

The right pane shows the code for 'routes/web.php':

```
| contains the "web" middleware group. Now create something great!  
|  
*/  
  
Route::get('home', [UserController::class, 'home'])->name('Home');  
Route::get('about', [UserController::class, 'about'])->name('about');  
Route::get('contact', [UserController::class, 'contact'])->name('contact');  
Route::get('/games', [UserController::class, 'index'])->name('games');
```

The screenshot shows a code editor with two tabs open: `userController.php` and `games.blade.php`. The `userController.php` tab contains the following PHP code:

```
<?php  
namespace App\Http\Controllers;  
  
use Illuminate\Http\Request;  
use App\Models\games ;  
  
class userController extends Controller  
{  
    public function index()  
    {  
        $games= games::all() ;  
  
        return view( view: 'games' , [  
            'games' => $games  
        ]) ;  
    }  
}
```

The `games.blade.php` tab contains the following Blade template code:

```
@extends('layout.app')  
  
@section('content')  
    <div style="text-align: center;">  
        <h1>  
            @foreach( $games as $g )  
                <h3>{{ $g->name_of_game }}</h3>  
            @endforeach  
        </h1>  
    </div>  
  
@endsection
```

The project structure on the left shows the following directory tree:

```
testLv  
  app  
    Console  
    Exceptions  
    Http  
    Models  
      games.php  
      postes.php  
      User.php  
    Providers  
    bootstrap  
    config  
    database  
    lang  
    node_modules library root  
    public  
  resources  
    css  
    js  
    views  
      layout  
      partial  
        navbar.blade.php  
        about.blade.php  
        contact.blade.php
```

The screenshot shows a code editor with two tabs open: `userController.php` and `games.blade.php`. The `userController.php` tab contains the same PHP code as the previous screenshot.

The `games.blade.php` tab contains the following Blade template code:

```
@extends('layout.app')  
  
@section('content')  
    <div style="text-align: center;">  
        <h1>  
            @foreach( $games as $g )  
                <h3>{{ $g->name_of_game }}</h3>  
            @endforeach  
        </h1>  
    </div>  
  
@endsection
```

The project structure on the left shows the same directory tree as the first screenshot.

**The condition of fetching datas from database :**

```

1 @extends('layout.app')
2
3 @section('content')
4
5     <div style="text-align: center;">
6         <h1>
7             @if($games->count() > 0 )
8                 @foreach( $games as $g )
9                     <h3>{{ $g->name_of_game }}</h3>
10                @endforeach
11            @else
12                <span>There is no game in the database !!!</span>
13            @endif

```

## How to fetch a data from database by id :

```
Route::get('uri: '/games/{id}' , [userController::class , 'show'] )->name( name: 'postShow' ) ;
```

```

1 @extends('layout.app')
2
3 @section('content')
4
5     <div style="...">
6         <h1>
7             @if($games->count() > 0 )
8                 @foreach( $games as $g )
9                     <h3><a href="{{ route('postShow' , ['id'=>$g->id]) }}>{{ $g->name_of_game }}</a></h3>
10                @endforeach
11            @else
12                <span>There is no game in the database !!!</span>
13            @endif
14        </h1>
15    </div>
16 @endsection

```

```

public function show($id)
{
    $games = games::find($id) ;

    return view( view: 'display',[

        'games' => $games
    ]);
}

```

```
1 @extends('layout.app')
2
3 @section('content')
4
5     <center>
6         <h1>{{ $games->name_of_game }}</h1>
7     </center>
8
9 @endsection
10
```

## How to fetch data by other column no an id :

```
16     ]
17 }
18
19 public function show($id)
20 {
21     $games = games::where('name_of_game' , '=' , 'GTA')->first();
22     dd($games);
23
24     return view( view: 'display',
25                 'games' => $games
26             );
27
28 }
```

## How to add something in database :

```
Route::get( uri: '/games/addGame' , [UserController::class , 'add'])->name( name: 'add-game') ;
```

```
Route::get( uri: '/games/addGame' , [UserController::class , 'add'])->name( name: 'add-game') ;
Route::post( uri: '/games/addGame' , [UserController::class , 'store'])->name( name: '[store-game') ;
```

The screenshot shows a code editor interface with a project structure on the left and a code editor on the right.

**Project Structure:**

- testLv
- app
- bootstrap
- config
- database
- lang
- node\_modules
- library root
- public
- resources

  - css
  - js
  - views
    - layout
    - partial
      - about.blade.php
      - addGame.blade.php
      - contact.blade.php
      - display.blade.php
      - games.blade.php
      - home.blade.php
    - routes
    - storage
  - tests
  - vendor
  - .editorconfig
  - .env
  - .env.example
  - .gitattributes

The screenshot shows a code editor interface with a project structure on the left and a code editor on the right.

**Project Structure:**

- web.php
- userController.php
- games.php
- addGame.blade.php

**Code Editor Content (games.php):**

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class games extends Model
9 {
10     use HasFactory;
11     protected $fillable = ['name_of_game' , 'language_of_game' , 'company_of_game' , 'prise_of_game' , 'num_of_version '];
12 }
13
```

```
UserController.php × games.php × addGame.blade.php ×
return view('view');
}

public function store(Request $request)
{
    $game = new games();
    $game->name_of_game = $request->nameOfGame;
    $game->language_of_game = $request->languageOfGame;
    $game->company_of_game = $request->companyOfGame;
    $game->prise_of_game = $request->priseOfGamef;
    $game->num_of_version = $request->numOfVersion;

    games::create([
        'name_of_game' => $request->nameOfGame ,
        'language_of_game' => $request->languageOfGame ,
        'company_of_game' => $request->companyOfGame ,
        'prise_of_game' => $request->priseOfGamef ,
        'num_of_version' => $request->numOfVersion
    ]);
    dd(...vars: 'game added successfully') ;
}
```

Update something in database :

```
11     {
12         $games= games::all() ;
13
14         return view( view: 'games' , [
15             'games' => $games
16         ] );
17     }
18
19     public function show($id)
20     {
21         $games = games::find(1) ;
22         $games->update([
23             'name_of_game' => 'new name of the game'
24         ]);
25
26         return view( view: 'display',[

27             'games' => $games
28         ]);
29     }
30 }
```

## Data base and Laravel

samedi 5 mars 2022

23:08

### Relation one to many :

Posts :

Id	Title	Content	Created_at	Updated_at
----	-------	---------	------------	------------

If we want add a comments , we should add ather table (comment ) because if we mae it as colomn in (postes table) we will have a lot of ligne with the same values exect the comment value , that s why we gonna make in a new table

Comments :

<b>Id</b>	<b>Content</b>	<b>Post_id</b>	<b>Created_at</b>	<b>Updated_at</b>
-----------	----------------	----------------	-------------------	-------------------

```

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

class CreateCommentsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('comments', function (Blueprint $table) {
            $table->id();
            $table->mediumText('content');
            $table->timestamps();
            $table->foreignId('post_id')->constrained();
        });
    }

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

```

**How to define a forieng key in database by the two lines above .**

```

namespace App\Models;

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

class posts extends Model
{
    use HasFactory;
    protected $fillable = ['title', 'post_content'];

    public function comments()
    {
        return $this->hasMany(comment::class); // because our posts has many comments
    }
}

```

**Because our posts has so many comment , at least one comment or 0 .**

The screenshot shows a code editor with a project structure on the left and a code file on the right. The project structure includes 'platformLaravel' with 'app', 'Http', 'Models', 'Providers', 'database', and 'migrations'. The 'comment.php' file under 'Models' is selected. The code in 'comment.php' is:

```
<?php  
namespace App\Models;  
use Illuminate\Database\Eloquent\Factories\HasFactory;  
use Illuminate\Database\Eloquent\Model;  
  
class comment extends Model  
{  
    use HasFactory;  
  
    public function post() // without 's' , because we reference on a post with lot of comments  
    {  
        return $this->belongsTo( related: posts::class );  
    }  
}
```

## How to display data :

The screenshot shows a code editor with a project structure on the left and a view file on the right. The project structure includes 'platformLaravel' with 'app', 'Http', 'Models', 'Providers', 'database', 'lang', 'public', and 'resources'. The 'comments.blade.php' file under 'views' is selected. The code in 'comments.blade.php' is:

```
@extends('layout.app')  
  
@section('content')  
    <div style="text-align: center;">  
        <br>  
        <h1 style="color: #dddddd">{{ $posts->title }}</h1>  
    </div>  
  
    <div style="text-align: center ; ">  
        <br>  
        @foreach($posts->comments as $c)  
            <h6 style="color: #ffffff">{{ $c->content }}</h6>  
        @endforeach  
    </div>  
@endsection
```

Or :

```

@extends('layout.app')

@section('content')
    <div style="text-align: center;">
        <br>
        <h1 style="color: #cccccc">{{ $post->title }}</h1>
    </div>

    <div style="text-align: center ; ">
        <br>
        @forelse($post->comments as $comment)
            <h6 style="color: #ffffcc">{{ $comment->content }}</h6>
        @empty
            <span style="color: #ffffcc">No comment yet</span>
        @endforelse
    </div>
@endsection

```

## Relation one to one:

Every post has one image and only one , so we need to create an other table image :

<b>Id</b>	<b>Path</b>	<b>Post_id</b>	<b>Created_at</b>	<b>Updated_at</b>
-----------	-------------	----------------	-------------------	-------------------

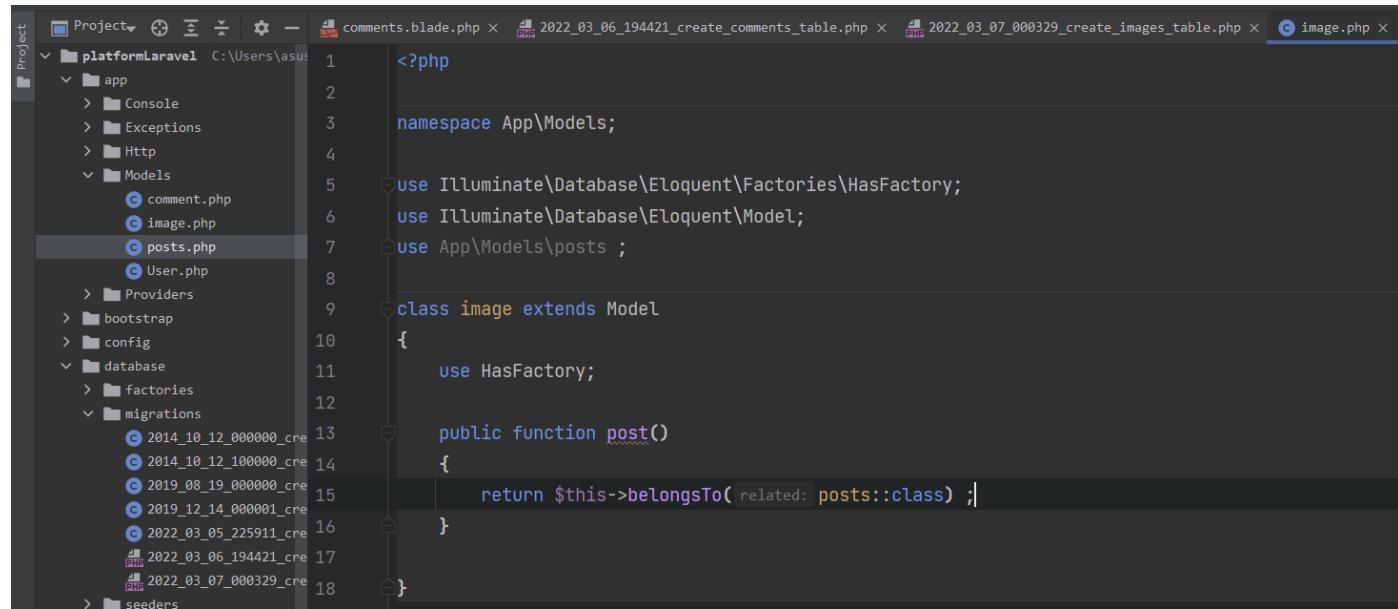
```

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

return new class extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('images', function (Blueprint $table) {
            $table->id();
            $table->string('path')->default('default.png');
            $table->foreignId('post_id')->constrained()->onDelete('cascade');
            $table->timestamps();
        });
    }
}

```

We first create our table and we add the 'foreign key for the' field ( See above ).



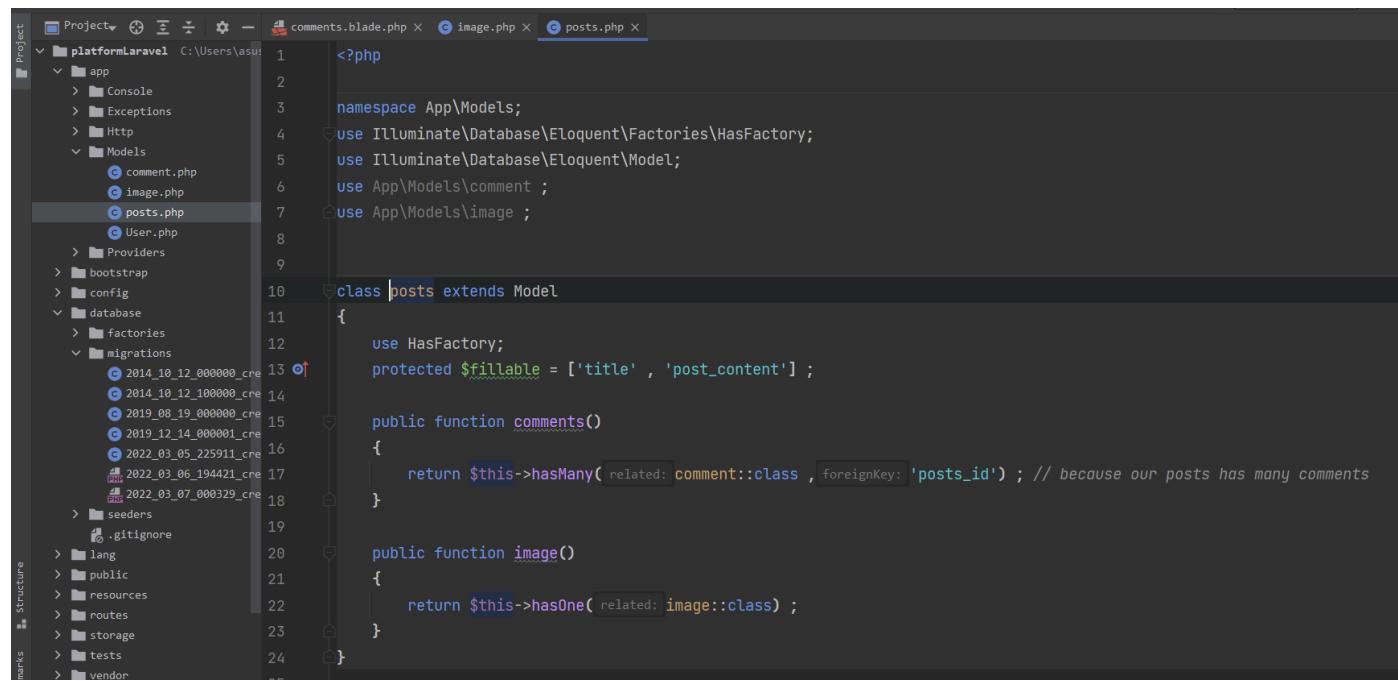
```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use App\Models\posts ;

class image extends Model
{
    use HasFactory;

    public function post()
    {
        return $this->belongsTo( related: posts::class ) ;
    }
}
```



```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use App\Models\comment ;
use App\Models\image ;

class posts extends Model
{
    use HasFactory;
    protected $fillable = ['title' , 'post_content'] ;

    public function comments()
    {
        return $this->hasMany( related: comment::class , [foreignKey: 'posts_id') ; // because our posts has many comments
    }

    public function image()
    {
        return $this->hasOne( related: image::class ) ;
    }
}
```

```

1  @extends('layout.app')
2
3  @section('content')
4
5      <div style="...">
6          <br>
7          <h1 style="...">
8              {{ $posts->title ?? 'None' }}
9          </h1>
10         <div>
11             
12         </div>
13
14

```

## Relation many to many :

A tags can be present a one post or lot of ones , but in the same time a post can have a lots of tags and it s the meaning of many to many relation .

We define a tag table as :

<b>Id</b>	<b>Name</b>	<b>Created_at</b>	<b>Updated_at</b>

We don't need here to define a foreing key , we need just to define a new table ( intermediate table ) this table wil containe bouth ids of 'posts' and 'tags' .

We will define it as : `post_tag`

<b>Post_id</b>	<b>Tag_id</b>	<b>Created_at</b>	<b>Updated_at</b>

And we will have an exaple like this :

<b>Post_id</b>	<b>Tag_id</b>
<b>1</b>	<b>3</b>
<b>1</b>	<b>2</b>
<b>2</b>	<b>5</b>
<b>2</b>	<b>2</b>
<b>3</b>	<b>5</b>

# Request

dimanche 17 avril 2022  
11:31

**La fonction 'input' dans l'objet 'request' :**  
**(allow to fetch the value from the key )**

The screenshot shows a code editor interface with three tabs at the top: 'testController.php' (active), '2022\_04\_17\_123447\_create\_acteurs\_table.php', and 'home.blade.php'. The main area displays a PHP controller class:

```
2
3     namespace App\Http\Controllers;
4     use App\Models\acteur;
5     use Illuminate\Http\Request;
6
7     class testController extends Controller
8     {
9
10        public function store(Request $request)
11        {
12
13            dd($request->input('key: 'name_of_acteur'));
14
15            acteur::create([
16                'nom' => $request->name_of_acteur,
17                'prenom' => $request->prenom_of_acteur,
18                'biographie' => $request->biographie
19            ]);
20        }
21    }
```

The code uses the Laravel framework, specifically the Request object to access input data. The 'store' method is defined within the 'testController' class, which extends the 'Controller' base class. The 'store' method takes a 'Request' object as a parameter and uses the 'dd' function to dump the input data. It then calls the 'create' method on the 'acteur' model, passing an array of key-value pairs extracted from the request input.

Return the path with the request object :

The screenshot shows a code editor with three tabs at the top: 'testController.php' (active), 'web.php', and 'home.blade.php'. The code in 'testController.php' is as follows:

```
4  use App\Models\acteur;
5  use Illuminate\Http\Request;
6
7  class testController extends Controller
8  {
9
10 public function store(Request $request)
11 {
12
13     dd($request->path());
14
15     acteur::create([
16         'nom' => $request->name_of_acteur,
17         'prenom'=> $request->prenom_of_acteur ,
18         'biographie' => $request->biographie
19     ]);
20 }
21 }
```

Verfy if we are in specifique URL :

```

class testController extends Controller
{
    public function store(Request $request)
    {
        dd($request->is( ...patterns: '/') ) ;

        acteur::create([
            'nom' => $request->name_of_acteur,
            'prenom'=> $request->prenom_of_acteur ,
            'biographie' => $request->biographie
        ]);
    }
}

```

Verify if we are in a specific route :

```

Route::get( uri: '/' , [testController::class, 'home'])->name( name: 'home') ;
Route::post( uri: '/store' , [testController::class , 'store'])->name( name: 'store_acteur') ;

public function store(Request $request)
{
    dd($request->routeIs( ...patterns: 'store_acteur')) ;

    acteur::create([
        'nom' => $request->name_of_acteur,
        'prenom'=> $request->prenom_of_acteur ,
        'biographie' => $request->biographie
    ]);
}

```

How to get the url :

```
public function store(Request $request)
{
    dd($request->url());
    acteur::create([
        'nom' => $request->name_of_acteur,
        'prenom'=> $request->prenom_of_acteur,
        'biographie' => $request->biographie
    ]);
}
```

How to know if the method is post ou get :

```
dd($request->isMethod('post'));
acteur::create([
    'nom' => $request->name_of_acteur,
    'prenom'=> $request->prenom_of_acteur,
    'biographie' => $request->biographie
]);
```

And how to return the method used :

```
{
    dd($request->method());
    acteur::create([
        'nom' => $request->name_of_acteur,
        'prenom'=> $request->prenom_of_acteur,
        'biographie' => $request->biographie
    ]);
}
```

## How to know the possible input :

```
dd($request->all());  
  
acteur::create([  
    'nom' => $request->name_of_acteur,  
    'prenom' => $request->prenom_of_acteur,  
    'biographie' => $request->biographie  
]);  
}
```

```
^ array:4 [▼  
  "_token" => "hxDv1BLIOwjnkszFiWyaZSIZViIdFIyhLjBVSUJvb"  
  "name_of_acteur" => "achraf khabar"  
  "prenom_of_acteur" => "aoaud"  
  "biographie" => "hahhahahaha"  
]
```

! : for the \_token is a key generated randomly due to the @csrf .

## Validation of formulaire :

jeudi 21 avril 2022  
22:10

## Inserting null object in a form :

To ovoid this issue we neet a request function named 'validate' :

```

public function store(Request $request)
{
    $request->validate([
        'title' => 'required',
        'post_content' => 'required'
    ]);

    posts::create([
        'title' => $request->title ,
        'post_content' => $request->post_content
    ]);

}

```

## How to display the error message when the form submition failed :



The screenshot shows a code editor with two tabs open: `addPosts.blade.php` and `userController.php`. The `addPosts.blade.php` tab is active and displays the following code:

```

@extends('Layout.app')

@section('content')
    <section data-aos="fade-up-right" class="contact-clean">
        @if( $errors->any())
            @foreach($errors->all() as $E)
                <center>
                    <span style="color: red">{{ $E }}</span> <br><br>
                </center>
            @endforeach
        @endif
        <form method="post" action="{{route('store-posts')}}"><h2 class="text-center">Add post</h2>
            @csrf
            <div class="mb-3">
                <input class="form-control" type="text" name="title" placeholder="title">
            </div>
            <div class="mb-3"></div>
            <div class="mb-3"><textarea class="form-control" name="post_content" placeholder="content" rows="14">
            </textarea>
            </div>
            <div class="mb-3">
                <button class="btn btn-primary" type="submit">send</button>
            </div>
        </form>
    </section>
@endsection

```

The code uses Blade templating syntax to check if there are any validation errors. If errors exist, it loops through them and displays each one in a red `<span>` element centered on the page. The form itself contains fields for the post title and content, and a submit button.

If we need that the input field needs at least 8 characters :

```
}

public function store(Request $request)
{
    $request->validate([
        'title' => 'required|min:8' ,
        'post_content' => 'required'
    ]);

    posts::create ([
        'title' => $request->title ,
        'post_content' => $request->post_content
    ]);

}
```

Other rules :

```
public function store(Request $request)
{
    $request->validate([
        'title' => 'required|min:8|max:255|unique:posts',
        'post_content' => 'required'
    ]);

    posts::create ([
        'title' => $request->title ,
        'post_content' => $request->post_content
    ]);
}
```

How to define a new rule :

```
PS C:\Users\asus\Desktop\install\htdocs\platformLaravel> php artisan make:rule Uppercase
Rule created successfully.
```

The screenshot shows a code editor interface with the following details:

- Project:** platformLaravel
- File:** Uppercase.php
- Code Content:**

```
15     {
16         //
17     }
18
19     /**
20      * Determine if the validation rule passes.
21      *
22      * @param string $attribute
23      * @param mixed $value
24      * @return bool
25     */
26     public function passes($attribute, $value)
27     {
28         return strtoupper($value) === $value ;
29     }
30
31     /**
32      * Get the validation error message.
33      *
34      * @return string
35     */
36     public function message()
37     {
38         return 'the title is not in uppercase form';
39     }
40 }
```
- Structure View:** Shows the directory structure of the project, including app, config, database, lang, public, resources, routes, storage, tests, vendor, .env, .env.example, .gitattributes, .gitignore, .styleci.yml, artisan, composer.json, composer.lock, package.json, phpunit.xml, README.md, and webpack.mix.js.

```
public function store(Request $request)
{
    $request->validate([
        'title' => ['required' , 'max:255' , 'min:8' , 'unique:posts' , new Uppercase],
        'post_content' => 'required'
    ]);

    posts::create([
        'title' => $request->title ,
        'post_content' => $request->post_content
    ]);
}
```

# Files

vendredi 22 avril 2022

23:09

## Forme update :

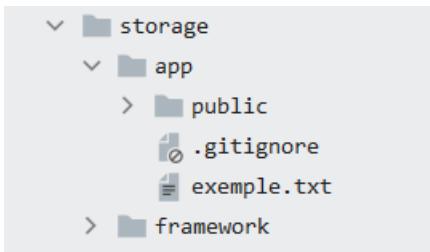
```
@endif
<form method="post" action="{{route('store-posts')}}" enctype="multipart/form-data"><h2 class="text-center">
    @csrf
    <div class="mb-3">
        <input class="form-control" type="text" name="title" placeholder="title">
    </div>
    <div class="mb-3"></div>
    <div class="mb-3"><textarea class="form-control" name="post_content" placeholder="content" rows="14">
        </textarea>
    </div>
    <div class="mb-3">
        <input class="form-control" type="file" name="title" placeholder="title">
    </div>
    <div class="mb-3">
        <button class="btn btn-primary" type="submit">send</button>
    </div>
```

## Storage in the local disk :

```
public function store(Request $request)
{
    storage::disk( name: 'local')->put( path: 'exemple.txt' , contents: 'My content') ;
    die();

    $request->validate([
        'title' => ['required' , 'max:255' , 'min:8' , 'unique:posts' , new Uppercase],
        'post_content' => 'required'
    ]);

    posts::create([
        'title' => $request->title ,
        'post_content' => $request->post_content
    ]);
}
```



## How to fetch a file from form uploaded from the pc :

```

@endif
<form method="post" action="{{route('store-posts')}}" enctype="multipart/form-data"><h2 class="text-center">Add post</h2>
    @csrf
    <div class="mb-3">
        <input class="form-control" type="text" name="title" placeholder="title">
    </div>
    <div class="mb-3"></div>
    <div class="mb-3"><textarea class="form-control" name="post_content" placeholder="content" rows="14">
        </textarea>
    </div>
    <div class="mb-3">
        <input class="form-control" type="file" name="photo_ashraf" placeholder="title">
    </div>
    <div class="mb-3">
        <button class="btn btn-primary" type="submit">send</button>
    </div>

```

```

public function store(Request $request)
{
    storage::disk('local')->put('photo_ashraf', $request->file('photo_ashraf')) ;
    die();
    $request->validate([
        'title' => ['required', 'max:255', 'min:8', 'unique:posts', new Uppercase],
        'post_content' => 'required'
    ]);

    posts::create([
        'title' => $request->title ,
        'post_content' => $request->post_content
    ]);
}

```



## Verify if a file exists or not :

```
public function store(Request $request)
{
    $name = storage::disk( name: 'local')->put( path: 'photo_ashraf' , $request->file( key: 'photo_ashraf')) ;
    dd(storage::disk( name: 'local')->exists($name)) ;
    $request->validate([
        'title' => ['required' , 'max:255' , 'min:8' , 'unique:posts' , new Uppercase],
        'post_content' => 'required'
    ]);

    posts::create ([
        'title' => $request->title ,
        'post_content' => $request->post_content
    ]);
}

public function store(Request $request)
{
    $name = storage::disk( name: 'local')->put( path: 'photo_ashraf' , $request->file( key: 'photo_ashraf')) ;
    dd(storage::disk( name: 'local')->missing($name)) ;
    $request->validate([
        'title' => ['required' , 'max:255' , 'min:8' , 'unique:posts' , new Uppercase],
        'post_content' => 'required'
    ]);

    posts::create ([
        'title' => $request->title ,
        'post_content' => $request->post_content
    ]);
}
```

## How to give the user the right to download a file :

```

public function store(Request $request)
{
    $name = storage::disk('local')->put('photo_ashraf' , $request->file('photo_ashraf')) ;
    return storage::download($name) ;
}

$request->validate([
    'title' => ['required' , 'max:255' , 'min:8' , 'unique:posts' , new Uppercase],
    'post_content' => 'required'
]);

posts::create([
    'title' => $request->title ,
    'post_content' => $request->post_content
]);

```

## Renvoyer la taille du fichier :

```

public function store(Request $request)
{
    $name = storage::disk('local')->put('photo_ashraf' , $request->file('photo_ashraf')) ;
    dd(storage::size($name)) ;

    $request->validate([
        'title' => ['required' , 'max:255' , 'min:8' , 'unique:posts' , new Uppercase],
        'post_content' => 'required'
    ]);
}

```

## How to store a file in the disk :

```

public function store(Request $request)
{
    // $name = storage::disk('local')->put('photo_ashraf' , $request->file('photo_ashraf')) ;
    dd(storage::put('fichier_ashraf' , $request->photo_ashraf));

    $request->validate([
        'title' => ['required' , 'max:255' , 'min:8' , 'unique:posts' , new Uppercase],
        'post_content' => 'required'
    ]);

    posts::create([
        'title' => $request->title ,
        'post_content' => $request->post_content
    ]);
}

```

## If we want the file as unique :

```
$path = $request->file('avatar')->storeAs(
    'avatars', $request->user()->id
);
```

## How to generate a random files :

```
public function store(Request $request)
{
    $filename = time().'.'.$request->file( key: 'photo_ashraf')->extension() ;

    dd($filename) ;
    $name = storage::disk('local')->put('photo_ashraf' , $request->file('photo_ashraf')) ;
    dd(storage::put( path: 'fichier_ashraf' , $request->photo_ashraf)) ;

    $request->validate([
        'title' => ['required' , 'max:255' , 'min:8' , 'unique:posts' , new Uppercase],
        'post_content' => 'required'
    ]);

    posts::create ([
        'title' => $request->title ,
        'post_content' => $request->post_content
```

## How to store an image for each post :

```
public function store(Request $request)
{
    $request->validate([
        'title' => ['required' , 'max:255' , 'min:8' , 'unique:posts' , new Uppercase],
        'post_content' => 'required'
    ]);

    // crer un directory named photos dans publique avec une photo named randomilly xxxx.pnj .
    $filename = time().'.'.$request->file( key: 'photo_ashraf')->extension() ;
    $path =  $request->file( key: 'photo_ashraf')->storeAs(
        path: 'photos' ,
        $filename ,
        options: 'public'
    );

    $post = posts::create ([
        'title' => $request->title ,
        'post_content' => $request->post_content
    ]);

    $image = new Image();
    $image->path = $path;

    $post->image()->save($image);
}
```

```
PS C:\Users\asus\Desktop\install\htdocs\platformLaravel> php artisan storage:link
```

```

@extends('layouts.app')

@section('content')
    <h1>{{ $post→content }}</h1>
    {{-- <span>{{ $post→image ? $post→image→path : "pas d'image" }}</span> --}}
    
    <hr>
    @forelse($post→comments as $comment)
        <div>{{ $comment→content }} | crée le {{ $comment→created_at→format('d/m/Y') }}</div>
    @empty
        <span>Aucun commentaire pour ce post.</span>
    @endforelse
    <hr>
    @forelse($post→tags as $tag)
        <span>{{ $tag→name }}</span>
    @empty
        <span>Aucun tag pour ce post.</span>
    @endforelse
    <hr>

    {{-- <span>Nom de l'artiste de l'image : {{ $post→imageArtist→name }}</span> --}}

@endsection

```

## Authentification

samedi 23 avril 2022

12:18

### How to fix the npm run dev problem :

<https://github.com/twbs/bootstrap/issues/36259>

- [autoprefixer: Replace color-adjust to print-color-adjust. · Issue #36259 · twbs/bootstrap \(github.com\)](#)
- [laravel - 1 WARNING in child compilations \(Use 'stats.children: true' resp. '--stats-children' for more details\) - Stack Overflow](#)

## 2 Answers

Sorted by: Highest score (default) ▾



<https://github.com/twbs/bootstrap/issues/36259> The color-adjust shorthand is currently deprecated and it depends on the autoprefixer@10.4.6.

13

I was able to fix this by reverting the autoprefixer package aswell as @Benno to version 10.4.5, run this:



`npm install autoprefixer@10.4.5 --save-exact`

*Npm install autoprefixer@10.4.5 --save-exact*

## Other npm run dev's issues :

```
\\"test-meme-library\\meme-library\\node_modules\\vue-loader\\lib\\plugin-webpack5.js',
\\"test-meme-library\\meme-library\\node_modules\\vue-loader\\lib\\plugin.js',
\\"test-meme-library\\meme-library\\node_modules\\vue-loader\\lib\\index.js',
\\"test-meme-library\\meme-library\\node_modules\\laravel-mix\\src\\components\\Vue.js',
\\"test-meme-library\\meme-library\\node_modules\\laravel-mix\\src\\components\\ComponentReg
\\"test-meme-library\\meme-library\\node_modules\\laravel-mix\\src\\Mix.js',
\\"test-meme-library\\meme-library\\node_modules\\laravel-mix\\setup\\webpack.config.js',
\\"test-meme-library\\meme-library\\node_modules\\webpack-cli\\lib\\webpack-cli.js',
\\"test-meme-library\\meme-library\\node_modules\\webpack-cli\\lib\\bootstrap.js',
\\"test-meme-library\\meme-library\\node_modules\\webpack-cli\\bin\\cli.js',
\\"test-meme-library\\meme-library\\node_modules\\webpack\\bin\\webpack.js'
```



You need to update your vue-loader

169

`npm update vue-loader`



And if it is not installed, install it



`npm i vue-loader`



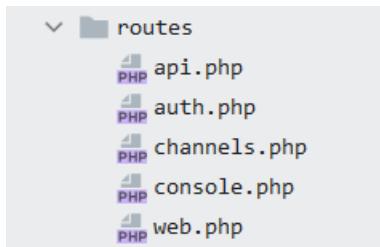
## We need to install the laravel breez if we need to have login properties :

```
PS C:\Users\asus\Desktop\install\htdocs\AuthonFication> composer require laravel/breeze --dev
```

```
Please execute the "npm install && npm run dev" command to build your assets.
```

```
PS C:\Users\asus\Desktop\install\htdocs\AuthonFication> php artisan breeze:install
```

We notice that we have a new folder inside route named auth :



```
PS C:\laragon\www\laravel-authentication> npm install
```

```
PS C:\Users\asus\Desktop\install\htdocs\AuthonFication> npm run dev
```

The login system work perfectly .

We need now to make a controller test :

```
PS C:\Users\asus\Desktop\install\htdocs\AuthonFication> php artisan make:controller testController
```

```
use App\Http\Controllers\testController;
use Illuminate\Support\Facades\Route;

/*
|--------------------------------------------------------------------------
|
| 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('welcome');
});

Route::get('/foo' , [testController::class,'test'])->name( name: 'foo' );
Route::get('/bare' , [testController::class,'test'])->name( name: 'bare' );

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

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

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class testController extends Controller
{
    public function foo()
    {
        return view( view: 'test.foo');
    }

    public function bare()
    {
        return view( view: 'test.bare');
    }
}
```

After creating these two views we need to make a middleware in order to verify some condition to allow acces to these two methods :

Now we gonna verify if the personn who want to get access to the two view is connected or not .

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class testController extends Controller
8 {
9     public function __construct()
10    {
11         $this->middleware( middleware: 'auth' ) ;
12    }
13
14    public function foo()
15    {
16        return view( view: 'test.foo' );
17    }
18
19    public function bare()
20    {
21        return view( view: 'test.bare' );
22    }
23
24 }
```

If we need filter the views which i can access without authentication :

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class testController extends Controller
{
    public function __construct()
    {
        $this->middleware( middleware: 'auth' )->except(['bare', 'foo']);
    }

    public function foo()
    {
        return view( view: 'test.foo');
    }

    public function bare()
    {
        return view( view: 'test.bare');
    }
}
```

If we need display something just only if i m connected :

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport"
6       content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
7     <meta http-equiv="X-UA-Compatible" content="ie=edge">
8     <title>Document</title>
9   </head>
10  <body>
11    <center>
12      <h1>
13        bare
14        @auth()
15          <span>
16            You are connected !!!
17          </span>
18        @endauth
19      </h1>
20    </center>
21  </body>
22</html>
```

With `@auth` i can display the name of the user and his informations  
:

The screenshot shows a code editor with a sidebar containing a project structure:

```

> public
<-- resources
    > css
    > js
    <-- views
        > auth
        > components
        > layouts
        <-- test
            bare.blade.php
            foo.blade.php
            dashboard.blade.php
            welcome.blade.php
        > routes
        > storage
        <-- tests
        > vendor
            .editorconfig
            .env
            .env.example
            .gitattributes
            .gitignore
            styleci.yml
            artisan
            composer.json
            composer.lock

```

The main pane displays a blade template with the following code:

```

7   <meta http-equiv="X-UA-Compatible" content="ie=>
8   <title>Document</title>
9   </head>
10  <body>
11  <center>
12  <h1>
13  bare
14  </h1>
15  @auth()
16  <span>
17  <br><br>
18  {{ Auth::user()->name }}
19  You are connected !!!
20  </span>
21  @endauth
22
23  </center>
24  </body>
25  </html>

```

I can do the same thing if i want to display something if we are only guest :

@guest :

Autorisation gates :

```
| PS C:\Users\asus\Desktop\install\htdocs\AuthonFication> php artisan make:migration add_admin_column_to_users_table --table=users
```

```
/**  
 * Run the migrations.  
 *  
 * @return void  
 */  
public function up()  
{  
    Schema::table( table: 'users', function (Blueprint $table) {  
        $table->boolean( column: 'admin')->default( value: 0) ; |  
    });  
}  
  
/**  
 * Reverse the migrations.  
 *
```

The screenshot shows a code editor interface with a project navigation bar at the top. The project path is `C:\Users\...\\AuthonFication`. The current file is `AuthServiceProvider.php`, which is part of the `App\Providers` directory. The code editor displays the following PHP code:

```
namespace App\Providers;

use ...

class AuthServiceProvider extends ServiceProvider
{
    /**
     * The policy mappings for the application.
     *
     * @var array<class-string, class-string>
     */
    protected $policies = [
        // 'App\Models\Model' => 'App\Policies\ModelPolicy',
    ];

    /**
     * Register any authentication / authorization services.
     *
     * @return void
     */
    public function boot()
    {
        $this->registerPolicies();
    }
}
```

The code editor has syntax highlighting and code completion features. The `AuthServiceProvider` class is highlighted in yellow, and the `use` statement is preceded by a tooltip icon. The `App\Providers` directory contains several other service provider classes like `AppServiceProvider`, `AuthServiceProvider`, `BroadcastServiceProvider`, `EventServiceProvider`, and `RouteServiceProvider`. The `tests` directory is also visible in the project structure.

The screenshot shows a code editor interface with the following details:

- Project Tree:** On the left, there's a tree view of the project structure. The 'AuthonFication' folder contains an 'app' folder which includes 'Console', 'Exceptions', 'Http', 'Models', 'Providers', 'View', and several service provider files: AppServiceProvider.php, AuthServiceProvider.php, BroadcastServiceProvider.php, EventServiceProvider.php, and RouteServiceProvider.php.
- Current File:** The file 'AuthServiceProvider.php' is open in the main editor area.
- Code Content:** The code defines an AuthServiceProvider class extending ServiceProvider. It includes policy mappings and a boot method that defines an 'access-admin' ability using the Gate facade.

```
use App\Models\User;
use Illuminate\Foundation\Support\Providers\AuthServiceProvider as ServiceProvider;
use Illuminate\Support\Facades\Gate;

class AuthServiceProvider extends ServiceProvider
{
    /**
     * The policy mappings for the application.
     *
     * @var array<class-string, class-string>
     */
    protected $policies = [
        'App\Models\Model' => 'App\Policies\ModelPolicy',
    ];

    /**
     * Register any authentication / authorization services.
     *
     * @return void
     */
    public function boot()
    {
        $this->registerPolicies();
        Gate::define('access-admin', function (User $user) {
            return $user->admin;
        });
    }
}
```

If we need to access to 'foo' :

The screenshot shows a code editor interface with a PHP file named `testController.php` open. The code defines a `testController` class that extends `Controller`. It contains two methods: `foo()` and `bare()`. The `foo()` method checks if the user has the `'access-admin'` ability using the `Gate::allows` method. If they do not have this ability, it aborts with a `403` status. Otherwise, it returns a view named `'test.foo'`. The `bare()` method simply returns a view named `'test.bare'`. The code editor includes syntax highlighting for PHP and shows the project structure on the left.

```
<?php

namespace App\Http\Controllers;

use Illuminate\Support\Facades\Gate;
use Illuminate\Http\Request;

class testController extends Controller
{
    public function __construct()
    {
        $this->middleware( middleware: 'auth' )->except( methods: 'bare' );
    }

    public function foo()
    {
        if (!Gate::allows( ability: 'access-admin' ) )
        {
            abort( code: '403' );
        }

        return view( view: 'test.foo' );
    }

    public function bare()
    {
        return view( view: 'test.bare' );
    }
}
```

## Emails

samedi 23 avril 2022

23:32

## HOW TO CONNECT YOUR DATABASE WITH YOUR PHP CODE :

Mail created successfully.

PS C:\Users\asus\Desktop\install\htdocs\AuthonFication> php artisan make:mail testMail

The screenshot shows a PHP development environment with the following details:

- Project:** AuthonFication (C:\Users\khabarachraf\Documents\GitHub\AuthonFication)
- File:** testMail.php
- Code Content:**

```
use Queueable, SerializesModels;

class testMail extends Mailable
{
    /**
     * Create a new message instance.
     *
     * @return void
     */
    public function __construct()
    {
        //
    }

    /**
     * Build the message.
     *
     * @return $this
     */
    public function build()
    {
        return $this
            ->from( address: 'khabarachraf@gmail.com')
            ->subject( subject: 'Mon object personnalisé')
            ->view( view: 'mail.test');
    }
}
```

The screenshot shows a PHP IDE interface with the following code in `testController.php`:

```

public function __construct()
{
    $this->middleware('auth')->except('methods: bare');
}

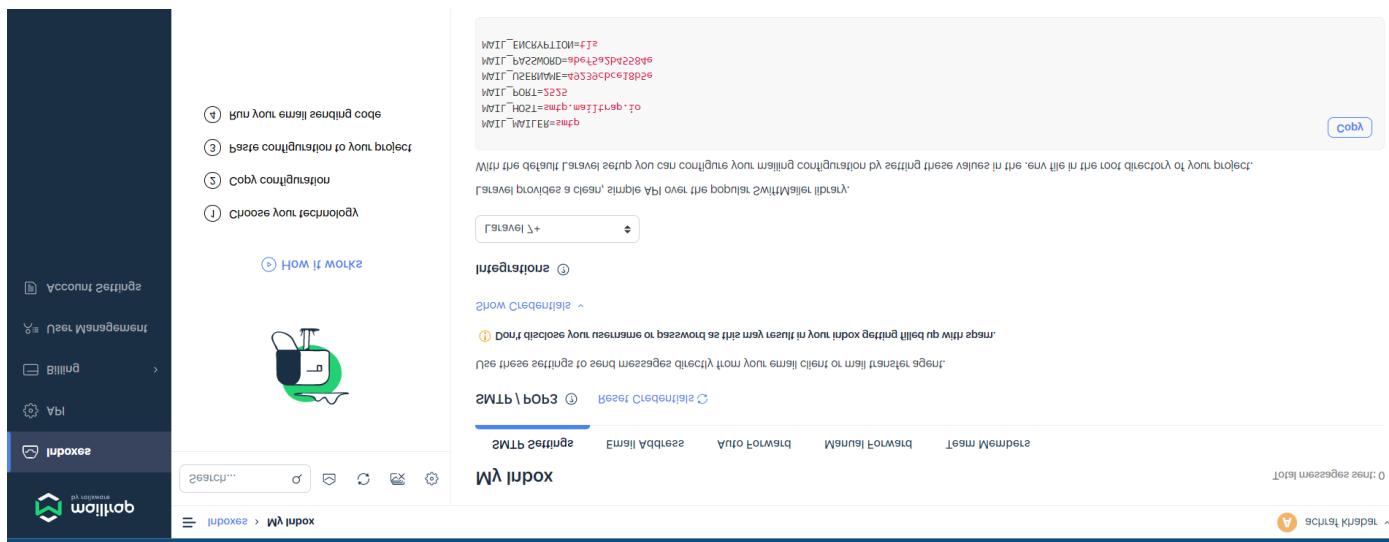
public function foo()
{
    if (!Gate::allows('ability: access-admin')) {
        abort(403);
    }

    return view('test.foo');
}

public function bare()
{
    Mail::to('users@test@mail.com')->send(new testMail());
    return view('test.bare');
}

```

The project structure on the left includes `AuthonFication`, `app`, `Http`, `Mail`, `Models`, `Providers`, `View`, `bootstrap`, `config`, `database`, `lang`, `node_modules`, `public`, and `resources`.



The screenshot shows the Mailtrap dashboard with the following details:

- Inboxes** section: "Inbox" selected.
- Mon object personnalisé** email received from "achraf khabar" at "khabarchraf@gmail.com" to "test@mail.com" a minute ago.
- Message Headers**:
 

```

From: <khabarchraf@gmail.com>
To: <test@mail.com>

```
- Message Content** (HTML tab):
 

My G-mail
- Message Options**:
  - HTML
  - HTML Source
  - Text
  - Raw
  - Spam Analysis
  - HTML Check (red)
  - Tech Info

## How to make a dynamic mail :



The screenshot shows a code editor with three tabs at the top: 'testMail.php', 'test.blade.php', and 'testController.php'. The 'testController.php' tab is active, displaying the following PHP code:

```
Project testMail.php test.blade.php testController.php
21
22     }
23
24     return view( view: 'test.foo');
25 }
26
27 public function bare()
28 {
29     $user = [ 'mail' => 'user@test.com' , 'name' => 'sami aouad' ] ;
30
31     Mail::to($user['mail'])->send(new testMail($user)) ;
32     return view( view: 'test.bare');
33 }
34 }
```

The screenshot shows a code editor interface with a sidebar containing project navigation (Project, Structure, Bookmarks) and tabs at the top (testMail.php, test.blade.php, testController.php). The main area displays the following PHP code:

```
4
5     use App\Models\User;
6     use Illuminate\Bus\Queueable;
7     use Illuminate\Contracts\Queue\ShouldQueue;
8     use Illuminate\Mail\Mailable;
9     use Illuminate\Queue\SerializesModels;
10    use PhpParser\Node\Expr\Array_;
11
12 class testMail extends Mailable
13 {
14     use Queueable, SerializesModels;
15
16     public $data = [] ;
17
18     public function __construct(Array $user)
19     {
20         $this->data = $user ;
21     }
22
23     public function build()
24     {
25         return $this
26             ->from( address: 'khabarachraf@gmail.com')
27             ->subject( subject: 'Mon object personnalisé')
28             ->view( view: 'mail.test');
29     }
30 }
```

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
        content="width=device-width, user-scalable=no
        <meta http-equiv="X-UA-Compatible" content="ie=edge
        <title>Document</title>
    </head>
    <body>
        <center>
            <h1>
                Bonjour Monsieur {{ $data['name'] }}
            </h1>
        </center>
    </body>
</html>
```

Inboxes > My Inbox > Mon object personnalisé

achraf khabar

2022-04-26 06:52, 734 Bytes

Mon object personnalisé

From: <khabarachraf@gmail.com>  
to: <user@test.com>  
Mon object personnalisé  
to: <test@mail.com> a minute ago

Show Headers

HTML    HTML Source    Text    Raw    Spam Analysis    HTML Check    Tech Info

Bonjour Monsieur sami aouad

Attachments :

```
testMail.php x test.blade.php x testController.php x
use Illuminate\Queue\SerializesModels;
use PhpParser\Node\Expr\Array_;

class testMail extends Mailable
{
    use Queueable, SerializesModels;

    public $data = [];

    public function __construct(Array $user)
    {
        $this->data = $user ;
    }

    public function build()
    {
        return $this
            ->from( address: 'khabarachraf@gmail.com')
            ->subject( subject: 'Mon object personnalisé')
            ->view( view: 'mail.test')
            ->attach(public_path( path: 'img/oracle.png'));
    }
}
```

## Generating mails with the mark down :

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6

PS C:\Users\asus\Desktop\install\htdocs\AuthonFication> php artisan make:mail testMarkdownMail -m mail.markdown-test
```

The screenshot shows a PHP IDE interface with a project structure on the left and a code editor on the right. The project structure for 'Authonfication' includes 'app', 'bootstrap', 'config', 'database', 'lang', 'node\_modules' (library root), 'public', 'resources' (containing 'css', 'js', 'views' with 'auth', 'components', 'layouts', 'mail' (containing 'markdown-test.blade.php', 'test.blade.php'), 'test', 'dashboard.blade.php', 'welcome.blade.php'), and 'routes'. The code editor displays a blade.php template:

```
1 @component('mail::message')
2 # Introduction
3
4 Bonjour sami aouad .
5
6
7 @component('mail::button', ['url' => $url])
8 Click here to see the video .
9 @endcomponent
10
11 Thanks,<br>
12 {{ config('app.name') }}
13 @endcomponent
14
```

The screenshot shows a PHP IDE interface with a project structure on the left and a code editor on the right. The project structure for 'Authonfication' includes 'app' (with 'Console', 'Exceptions', 'Http', 'Mail' (containing 'testMail.php', 'testMarkdownMail.php'), 'Models', 'Providers', 'View'), 'bootstrap', 'config', 'database', 'lang', 'node\_modules' (library root), 'public', 'resources', 'routes', 'storage', 'tests' (highlighted in green), 'vendor' (with '.editorconfig', '.env', '.env.example', '.gitattributes', '.gitignore', '.styleci.yml', 'artisan'), and 'tests'. The code editor displays a testMarkdownMail.php file:

```
1 <?php
2
3 namespace App\Mail;
4
5 use ...
6
7 class testMarkdownMail extends Mailable
8 {
9     use Queueable, SerializesModels;
10    public $url = 'https://www.youtube.com/watch?v=RLGWYpY07kg&list=PLeeuvNW2FHVj4vHJRj9UDeDsXshHlnHJk&index=20';
11
12    public function __construct()
13    {
14        //
15    }
16
17    public function build()
18    {
19        return $this->markdown('mail.markdown-test');
20    }
21
22}
```

The screenshot shows a PHP IDE interface with the following details:

- Project:** AuthonFication
- File:** testController.php
- Code Content:**

```
11 class testController extends Controller
12 {
13     public function __construct()
14     {
15         $this->middleware( middleware: 'auth' )->except( methods: 'bare' );
16     }
17
18     public function foo()
19     {
20         if (!Gate::allows( ability: 'access-admin' ) )
21         {
22             abort( code: '403' );
23         }
24
25         return view( view: 'test.foo' );
26     }
27
28     public function bare()
29     {
30         // $user = ['mail' => 'user@test.com' , 'name' => 'sami aouad'];
31
32         Mail::to( users: 'user@test.com' )->send( new testMarkdownMail() );
33
34         return view( view: 'test.bare' );
35     }
}
```

- Project Structure:** Shows the directory structure of the project, including app, Http, Mail, and tests.
- Toolbars:** Standard IDE toolbars for file operations.

## Query builder

vendredi 29 avril 2022  
00:45

**Fetch all data from data base :**

The screenshot shows a PHP development environment with the following details:

- File Menu:** File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help.
- Project:** projectGestionEtudiant
- Code Editor:** The file `filiere.blade.php` is open, showing PHP code for a controller. The code includes methods `destroy()`, `filiere()`, and `delete($id)`. It uses Laravel's `Auth::guard('admin')` for authentication and `DB::table('users')` for querying the database.
- Database:** A sidebar titled "Database" shows the structure of the `gestionetudiantdb` database, specifically the `users` table. The table has columns: id, filiere\_id, name, email, password, remember\_token, created\_at, and updated\_at. It also lists failed jobs and migrations.
- Terminal:** Shows the command `@localhost: users synchronized (750 ms) (yesterday 17:46)`.
- Version Control:** Shows the path `\App\Http\Controllers\Admin\ adminController > filiere()`.
- Event Log:** Shows PHP version information: PHP: 8.0 44:52 LF UTF-8 4 spaces.

Use a query with where condition :

The screenshot shows a code editor with three tabs at the top: 'filiere.blade.php' (file icon), 'adminController.php' (blue circle icon), and 'admin.php' (PHP icon). The 'adminController.php' tab is active. The code is as follows:

```
29     public function destroy()
30     {
31         Auth::guard('name: 'admin')->logout() ;
32         return redirect( to: '/') ;
33     }
34
35     public function filiere()
36     {
37         // $students = User::all() ;
38         //
39         // return View('filiere' , [
40         //     'students' => $students
41         //]);
42
43         $data = DB::table( table: 'users')
44             ->where( column: 'id' , operator: 1) |
45             ->get() ;
46
47         return View( view: 'filiere',[ 'data' => $data]) ;
48     }
49 }
```

Find :

The screenshot shows a code editor interface with a dark theme. On the left, there's a vertical sidebar with tabs for 'Project', 'Structure', 'Bookmarks', and 'npm'. The main area displays a PHP file named 'adminController.php'.

```
filiere.blade.php × adminController.php × admin.php ×
Project
Structure
Bookmarks
npm
29     public function destroy()
30     {
31         Auth::guard('name: 'admin')->logout();
32         return redirect(to: '/');
33     }
34
35     public function filiere()
36     {
37         // $students = User::all();
38         //
39         // return View('filiere', [
40         //     'students' => $students
41         // ]);
42
43         $data = DB::table('users')->find(1);
44
45         return View('filiere', ['data' => $data]);
46     }
47
48     public function delete($id)
49     {
50         $student = User::find($id);
51         $student->delete();
52         return redirect(to: 'admin/filiere');
53     }
54
55 }
```

Count :

The screenshot shows a code editor interface with a dark theme. On the left, there's a sidebar with tabs for 'Project', 'Structure', 'Bookmarks', and 'npm'. The main area displays a PHP file with the following code:

```
filiere.blade.php x adminController.php x admin.php x
public function destroy()
{
    Auth::guard('name: 'admin')->logout();
    return redirect(to: '/');
}

public function filiere()
{
    $students = User::all();
    return View('filiere' , [
        'students' => $students
    ]);
}

$data = DB::table('users')->count('id');

return View('filiere',[ 'data' => $data]);
}

public function delete($id)
{
    $student = User::find($id);
    $student->delete();
    return redirect(to: 'admin/filiere');
}

}
```

Insert :

The screenshot shows a code editor with three tabs at the top: 'filiere.blade.php' (file icon), 'adminController.php' (PHP icon, currently selected), and 'admin.php' (file icon). The main area displays the contents of the 'adminController.php' file.

```
filiere.blade.php x adminController.php x admin.php x
29     public function destroy()
30     {
31         Auth::guard( name: 'admin')->logout() ;
32         return redirect( to: '/') ;
33     }
34
35     public function filiere()
36     {
37         // $students = User::all() ;
38         //
39         //
40         //
41         // $data = DB::table( table: 'users')
42         //             ->insert([
43         //                 'name' => 'sami aouad',
44         //                 'email' => 'samiaouad@gmail.com' ,
45         //             ]) ;
46
47         return View( view: 'filiere',[ 'data' => $data]) ;
48
49     }
50
51     public function delete($id)
52     {
53         $student = User::find($id) ;
54         $student->delete() ;
55     }

```

Update :

projectGestionEtudiant > app > Http > Controllers > admin > adminController.php > adminController.php

```
Project filiere.blade.php × adminController.php × admin.php ×
29     public function destroy()
30     {
31         Auth::guard('name: admin')->logout();
32         return redirect(to: '/');
33     }
34
35     public function filiere()
36     {
37         // $students = User::all();
38         //
39         // return View('filiere', [
40         //     'students' => $students
41         // ]);
42
43         $data = DB::table(table: 'users')
44             ->where(column: 'id', operator: 6)
45             ->update([
46                 'name' => 'sami aouad',
47                 'email' => 'samiaouad@gmail.com'
48             ]);
49
50         return View(view: 'filiere', ['data' => $data]);
51     }
52
53     public function delete($id)
54     {
55         $student = User::find($id);
```

Delete :

The screenshot shows a code editor interface with several tabs at the top: 'filiere.blade.php' (highlighted), 'adminController.php', and 'admin.php'. The main area displays PHP code for an 'AdminController' class. The code includes methods for destroying and listing students, and a method for deleting a student by ID. It also includes a comment block and a view return statement.

```
projectGestionEtudiant > app > Http > Controllers > admin > C adminController.php > C adminCo
Project filiere.blade.php × C adminController.php × PHP admin.php ×
29     public function destroy()
30     {
31         Auth::guard(name: 'admin')->logout() ;
32         return redirect(to: '/') ;
33     }
34
35     public function filiere()
36     {
37         // $students = User::all() ;
38         //
39         // return View('filiere' , [
40         //     'students' => $students
41         // ]);
42
43         $data = DB::table(table: 'users')
44             ->where(column: 'id', operator: 22)
45             ->delete() ;
46
47         return View(view: 'filiere',[ 'data' => $data]) ;
48     }
49
50     public function delete($id)
51     {
52         $student = User::find($id) ;
53         $student->delete() ;
54         return redirect(to: 'admin/filiere') ;
55     }

```

Join :

The screenshot shows a code editor interface with three tabs at the top: 'filiere.blade.php', 'adminController.php', and 'admin.php'. The 'adminController.php' tab is active and displays the following PHP code:

```
29     public function destroy()
30     {
31         Auth::guard( name: 'admin')->logout() ;
32         return redirect( to: '/') ;
33     }
34
35     public function filiere()
36     {
37         ////
38         ////
39         ////
40         ////
41         ////
42
43         $data = DB::table( table: 'users')
44             ->join( table: 'filiere', first: 'users.filieres_id', operator: '=', second: 'filiers.id')
45             ->get();
46
47         return View( view: 'filiere',[ 'data' => $data]) ;
48     }
49
50     public function delete($id)
51     {
52         $student = User::find($id) ;
53         $student->delete() ;
54         return redirect( to: 'admin/filiere') ;
55     }

```

Get data from a specific table :

The screenshot shows a code editor with three tabs open: 'filiere.blade.php', 'adminController.php', and 'admin.php'. The 'adminController.php' tab is active and displays the following PHP code:

```
projectGestionEtudiant > app > Http > Controllers > admin > C adminController.php > C adminController > m liere
Project filiere.blade.php × C adminController.php × PHP admin.php ×
29     public function destroy()
30     {
31         Auth::guard( name: 'admin')->logout() ;
32         return redirect( to: '/') ;
33     }
34
35     public function filiere()
36     {
37         $students = User::all() ;
38
39         // return View('filiere' , [
40         //     'students' => $students
41         // ]);
42
43         $data = DB::table( table: 'users')
44             ->join( table: 'filiere', first: 'users.filieres_id', operator: '=', second: 'filiers.id')
45             ->select( columns: 'users.*') // or 'filieres.*'
46             ->get( );
47
48         return View( view: 'filiere',[ 'data' => $data]) ;
49     }
50
51     public function delete($id)
52     {
53         $student = User::find($id) ;
54         $student->delete() ;
55         return redirect( to: 'admin/filiere') ;
```

Add condition in join :

The screenshot shows a code editor interface with a dark theme. The left sidebar has sections for 'Project', 'Structure', 'Bookmarks', and 'PHP'. The main area displays the following PHP code:

```
projectGestionEtudiant > app > Http > Controllers > admin > C adminController.php > C adminController > m liere
filiere.blade.php <--> C adminController.php <--> PHP admin.php <--> liere

29     public function destroy()
30     {
31         Auth::guard( name: 'admin')->logout() ;
32         return redirect( to: '/') ;
33     }
34
35     public function filiere()
36     {
37         $students = User::all() ;
38
39         // return View('filiere' , [
40         //     'students' => $students
41         ]);|
42
43         $data = DB::table( table: 'users')
44             ->join( table: 'filiere', first: 'users.filières_id', operator: '=', second: 'filières.id')
45             ->select( columns: 'users.*') // or 'filières.*'
46             ->where( column: 'filières.label', operator: 'GINF1')
47             ->get( );
48
49         return View( view: 'filiere',[ 'data' => $data]) ;
50     }
51
52     public function delete($id)
53     {
54         $student = User::find($id) ;
55         $student->delete() ;
```

## Notifications

dimanche 1 mai 2022

00:39

How to create a notification :

```
PS C:\Users\asus\Desktop\project gestion etudiant laravel\back-end\projectGestionEtudiant> php artisan make:notification UserNotification
```

How to make a notification : ( we choose the model which we want it to be notifiable ) :

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help AuthonFication - RegisteredUserController.php - Administrator
AuthonFication > app > Http > Controllers > Auth > RegisteredUserController.php > RegisteredUserController > store
Project  RegisteredUserController.php x
34
35     public function store(Request $request)
36     {
37         $request->validate([
38             'name' => ['required', 'string', 'max:255'],
39             'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
40             'password' => ['required', 'confirmed', Rules\Password::defaults()],
41         ]);
42
43         $user = User::create([
44             'name' => $request->name,
45             'email' => $request->email,
46             'password' => Hash::make($request->password),
47         ]);
48
49         event(new Registered($user));
50
51         $user->notify( new UserNotification());
52
53         Auth::login($user);
54
55         return redirect(to: RouteServiceProvider::HOME);
56     }
57
58 }
```

Add something to the mail configuration :

```
46     'password' => Hash::make($request->password),
47 ];
48
49         event(new Registered($user));
50
51         $post = ['title' => 'super title'];
52         $user->notify( new UserNotification($user , $post));
53
54         Auth::login($user);
55
56         return redirect(to: RouteServiceProvider::HOME);
57
58 }
```

The screenshot shows a code editor interface with a dark theme. On the left, there's a vertical sidebar with icons for 'Project', 'Structure', 'Bookmarks', and 'npm'. The main area displays the code for the `UserNotification.php` file. The file is a PHP class definition:

```
9  
10 class UserNotification extends Notification  
11 {  
12     use Queueable;  
13     public $user;  
14     public $post;  
15     /**  
16      * Create a new notification instance.  
17      *  
18      * @return void  
19      */  
20     public function __construct($user, $post)  
21     {  
22         $this->user = $user;  
23         $this->post = $post;  
24     }  
25  
26     /**  
27      * Get the notification's delivery channels.  
28      *  
29      * @param mixed $notifiable  
30      * @return array  
31      */  
32     public function via($notifiable)  
33     {  
34         return ['mail'];  
35     }  
}
```

The screenshot shows a code editor with two tabs open: 'RegisteredUserController.php' and 'UserNotification.php'. The 'UserNotification.php' tab is active, displaying the following PHP code:

```
34     return ['mail'];
35 }
36
37 /**
38 * Get the mail representation of the notification.
39 *
40 * @param mixed $notifiable
41 * @return \Illuminate\Notifications\Messages\MailMessage
42 */
43 public function toMail($notifiable)
44 {
45     return (new MailMessage)
46         ->line('Notification for the user ' . $notifiable->name() . ' for the post ' . $this->post['title'])
47         ->action('Notification Action', url('/'))
48         ->line('Thank you for using our application!');
49 }
50
51 /**
52 * Get the array representation of the notification.
53 *
54 * @param mixed $notifiable
55 * @return array
56 */
57 public function toArray($notifiable)
58 {
59     return [
60         //
```

How to use our mail :

```
PS C:\Users\asus\Desktop\install\htdocs\AuthonFication> php artisan make:mail testNotifactionMail
```

The screenshot shows a code editor interface with a project navigation sidebar on the left and a main code editor area on the right.

**Project Structure:**

- Project
- Providers
- View
  - bootstrap
  - config
  - database
  - lang
  - node\_modules library root
  - public
  - resources
    - css
    - js
    - views
      - auth
      - components
      - layouts
      - mail
        - markdown-test.blade.php
        - test.blade.php
        - testMailNotifiacation.blade.php
  - test
    - dashboard.blade.php
    - welcome.blade.php
  - routes
  - storage
  - tests
  - vendor

.editorconfig  
.env  
.env.example  
.gitattributes  
.gitignore  
.styleci.yml  
artisan  
composer.json  
composer.lock

**Code Editor (testNotifactionMail.php):**

```
10 class testNotifactionMail extends Mailable
11 {
12     use Queueable, SerializesModels;
13     public $user ;
14     public $post ;
15     /**
16      * Create a new message instance.
17      *
18      * @return void
19     */
20     public function __construct($user , $post )
21     {
22         $this->user = $user ;
23         $this->post = $post ;
24     }
25     /**
26      * Build the message.
27      *
28      * @return $this
29     */
30     public function build()
31     {
32         return $this->view( view: 'mail.testMailNotifiacation' );
33     }
34 }
35 }
```

The screenshot shows a code editor interface with the following details:

- Project Structure:** The left sidebar displays the project structure of `AuthonFication`. It includes the `app`, `Notifications`, and `UserNotification` files.
- Code Editor:** The main area shows the `UserNotification.php` file. The code defines two methods:
  - `toMail()`: Returns an array with the key `'mail'`.
  - `toArray($notifiable)`: Returns an array representation of the notification.
- File Tabs:** The tabs at the top show `testNotifactionMail.php`, `testMailNotifiacation.blade.php`, and `UserNotification.php`.
- Bottom Status Bar:** The status bar indicates the current file path: `\App\Notifications > UserNotification > toMail()`.

## Notifications and database :

```
PS C:\Users\asus\Desktop\install\htdocs\AuthonFication> php artisan notifications:table
```

AuthonFication > app > Notifications > UserNotification.php > UserNotification > toArray

```
Project testNotifactionMail.php × testMailNotifiacation.blade.php × UserNotification.php ×
21     public function __construct(User $user, Post $post)
22     {
23         $this->user = $user;
24         $this->post = $post;
25     }
26
27     /**
28      * Get the notification's delivery channels.
29      *
30      * @param mixed $notifiable
31      * @return array
32      */
33     public function via($notifiable)
34     {
35         return ['mail', 'database'];
36     }
37
38     /**
39      * Get the mail representation of the notification.
40      *
41      * @param mixed $notifiable
42      * @return testNotifactionMail
43      */
44     public function toMail($notifiable)
45     {
46         return (new testNotifactionMail($notifiable, $this->post))
47             ->to($notifiable->email);
48     }

```

The screenshot shows a code editor interface with three tabs at the top: 'testNotifactionMail.php' (active), 'testMailNotifiacation.blade.php', and 'UserNotification.php'. The code in 'UserNotification.php' is as follows:

```
42     */
43     public function toMail($notifiable)
44     {
45         return (new testNotifactionMail($notifiable , $this->post))
46                 ->to($notifiable->email) ;
47     }
48
49
50     /**
51      * Get the array representation of the notification.
52      *
53      * @param mixed $notifiable
54      * @return array
55      */
56     public function toArray($notifiable)
57     {
58         return [
59             'title' => 'my title : '.$this->post['title'] ,
60             'my_email' => $notifiable->email|
61         ];
62     }
63 }
```

```

<x-app-layout>
    <x-slot name="header">
        <h2 class="font-semibold text-xl text-gray-800 leading-tight">
            {{ __('Dashboard') }}
        </h2>
    </x-slot>

    <div class="py-12">
        <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
            <div class="bg-white overflow-hidden shadow-sm sm:rounded-lg">
                <div class="p-6 bg-white border-b border-gray-200">
                    @foreach(auth()->user()->notifications as $notification)
                        <span>{{ $notification }}</span>
                    @endforeach
                </div>
            </div>
        </div>
    </div>
</x-app-layout>

```

```

<x-app-layout>
    <x-slot name="header">
        <h2 class="font-semibold text-xl text-gray-800 leading-tight">
            {{ __('Dashboard') }}
        </h2>
    </x-slot>

    <div class="py-12">
        <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
            <div class="bg-white overflow-hidden shadow-sm sm:rounded-lg">
                <div class="p-6 bg-white border-b border-gray-200">
                    @foreach(auth()->user()->notifications as $notification)
                        <span>{{ $notification->data['title'] }}</span>
                    @endforeach
                </div>
            </div>
        </div>
    </div>
</x-app-layout>

```

## Middleware

mardi 3 mai 2022

23:17

How to create our own middleware :

```
PS C:\Users\asus\Desktop\install\htdocs\AuthonFication> ph artisan make:middleware isAdmin
```

The screenshot shows a code editor with two tabs open: `Kernel.php` and `isAdmin.php`.

**Kernel.php:**

```
47 /**
48 * The application's route middleware.
49 */
50 /**
51 * These middleware may be assigned to groups or used individually.
52 */
53 /**
54 * @var array<string, class-string/string>
55 */
56 protected $routeMiddleware = [
57     'auth' => \App\Http\Middleware\Authenticate::class,
58     'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
59     'auth.session' => \Illuminate\Session\Middleware\AuthenticateSession::class,
60     'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
61     'can' => \Illuminate\Auth\Middleware\Authorize::class,
62     'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
63     'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,
64     'signed' => \Illuminate\Routing\Middleware\ValidateSignature::class,
65     'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
66     'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
67     'is.admin' => \App\Http\Middleware\isAdmin::class
68 ];
```

**isAdmin.php:**

```
8 class isAdmin
9 {
10     /**
11      * Handle an incoming request.
12      *
13      * @param \Illuminate\Http\Request $request
14      * @param \Closure(\Illuminate\Http\Request): (\Illuminate\Http\Response|\Illuminate\Http\RedirectResponse)
15      * @return \Illuminate\Http\Response|\Illuminate\Http\RedirectResponse
16     */
17     public function handle(Request $request, Closure $next)
18     {
19         if ( auth()->user()->admin == 1 )
20         {
21             return $next($request);
22         }else
23         {
24             abort(code: 403) ;
25         }
26     }
27 }
28 }
```