Note : Installer un project laravel  avec composer : composer create-project laravel/laravel laravelformation --prefer-dist ;

composer require doctrine/dbal

composer require laravel/breeze

php artisan breeze:install

\*\*\*

Controlleur :

-pour appeler une vue

controlleur

<?php

namespace App\Http\Controllers;

class PostController extends Controller{

   public function index(){

    return view('articles');

   }

}

Ou

En ligne de commande php artisan make:controller PostController //crere controller

-routes

Route::get('/','App\Http\Controllers\PostController@index' );

Ou

Route::get('/',[PostController::class,'index']);

Ou avec argument

Route::get('Post/{id}',[PostController::class,'show']);

Envoyer une donné à une vue

 $title='mon super titre';

        return view('articles',compact('title'));

ou

ou

 $posts = [

             'Mon super titre2',

            'mon super titre4'  ];

return view('articles',compact('posts'));

ou

return view('articles')->with('title',$title);

Affichage donné de la vue variable simple :

{{ $title }}

Affichage donné tableau

@foreach($posts as $post)

<h3>{{ $post }}</h3>

            @endforeach

Vues :

Voir dans layouts/app / pour la séparation des code html

Voir dans articles .blade.php // pour voir comment avoir des contenues dynamiques

<!DOCTYPE html>

<html>

    <head>

<title>Mon super titre</title>

    </head>

    <body>

        @include('partial.navbar')//pour inclure un menue

       @yield('content')

    </body>

</html>

@extends('layouts.app')

@section('content')

<h1>articles</h1>

@foreach($posts as $post)

<h3>{{ $post }}</h3>

            @endforeach

@endsection

-Pour les fichiers css et js //on install d’abord node js

npm run dev

npm run watch // pour sass dans video 5

npm install -d tailwindcss //install tailwindcss

npx tailwindcss init dans video 5 il choisit les fichoier css ou js etc pour notre app et pour le style css aussi

Link style css : {{ asset('css/app.css') }} // et javascript

C’est dans .env qu’on spécie les paramétres de la base de donné

-c’est dans le dossier database il y’a migration pour la création des tables

php artisan make:model Post -m //créer une modèle après on va

dans le model pour créer les colonnes de la table voir dans database migrations

et on ajoute ces lignes

$table->string('title');

            $table->mediumText('content');

-On peut aussi définir dans le model les clés primaires exemple :

Protected $primarykey=’flight\_id’ ;

MIGRATIONS

-php atisan migrate

Prérequis avant la commande migrate on modifie le fichier APP /provider/ AppServiceProvider :

<?php

namespace App\Providers;

use Illuminate\Support\ServiceProvider;

use Illuminate\Support\Facades\Schema;

class AppServiceProvider extends ServiceProvider

{

/\*\*

  \* Bootstrap any application services.

  \*

  \* @return void

  \*/

public function boot()

{

Schema::defaultStringLength(191);

}

/\*\*

  \* Register any application services.

  \*

  \* @return void

  \*/

public function register()

{

//

}

}

**Factories :**

pour générer des donnés pour insérer dans la base de donné

-création :

php artisan make:factory PostFactory --model=Post // le model est Post

puis dans PostFactory on ajoute dans la définition :

'content' => $this->faker->paragraph //content est un attribut de la table Post

php artisan tinker//dans le terminal pour écrire du php

Post::factory()->count(10)->create();//enregistrement de 10 lignes dans notre table Post

ORM :

Pour recupérer les données dans la bd d’une table on fait :

$posts = Post::all(); //Post est le modèle

$posts=Post ::findorfail($id) ;//pour un seul élément

Ou

$posts=Post::where ('title','=','Non id recusandae voluptates inventore cum et dicta.')->get() ;//avec where aulieu de get aussi ->first() ;pour la premiere enragistrement

Conditions dans une vue :

 @if ($posts->count() > 0)

@foreach($posts as $post)

<h3>{{ $post->title }}</h3>

            @endforeach

            @else

            <span> il n'y as pas encore d'enregistrement dans Post</span>

            @endif

-Dans un lien appeler une route avec un argument :

<h3><a href="{{ route('posts.show',['id'=>$post->id]) }}">{{ $post->title }}</a></h3>

**Formulaire dans le root on remplace le get par un Post :**

Exemple :

<form>

@csrf //pour une protection csrf

</form>

Route::post('create',[PostController::class,'store'])->name('posts.store')

Recuperation formulaire:

-on ajoute en argument dans notre méthode de recupération Request $request :

     public function store(Request $request){

        $post=new Post();//instanciation modéle

        $post->title=$request->title;//affectation attribute d’une donné formulaire

        $post->content=$request->content;

        $post->save();//enregistremnt dans la table post

dd($post);

    }

-pour insertion des donnés formulaire prérequis on ajoute dans le modèle protected $fillable= ['title','content'] ;

Ou

protected $guarded=[];

:

Post::create([

'title' => $request->title, //insertion $request permet de recupérer

'content' => $request->content

]);

-Pour Mis a jour table

     $post=Post::find(1);

        $post->update([

'title'=>'titre édité'

]);

-suppression donne table

 $post=Post::find(1);

   $post->delete();

\*Pour approfondire voir documentation laravel eloquent ou relation

**RELATION ONE TO MANY :**

-order by et filtre par N :

$posts= Post::orderBy('title')->take(3)->get();

-clé secondaire(voir dans migration de comment) ;

Schema::table('posts', function (Blueprint $table) {

$table->unsignedBigInteger('user\_id');

$table->foreign('user\_id')->references('id')->on('users');

});

Ou

  $table->foreignId('post\_id')->constrained()->onDelete('cascade');//ondelete permette si un Post est supprimer l’image associé aussi est supprimer

Relation one to many (1,\*) ;

-exemple: voir dans le modèle Post

public function comments(){

        return $this->hasmany(Comment::class);//comment c’est le nom du modéle

    }

-Pour appeler la function d’une modèle exemple comments (voir dans article.blade.php:

1 :

@foreach($post->comments as $comment)

<span> {{ $comment->content }}</span>

@endforeach

2 :Astuces si le tableau comment est vide vérification et formatage date:

@forelse($post->comments as $comment)

<span> {{ $comment->content }}| créer le {{ $comment->created\_at->format('d-m-y') }}</span>

@empty

<span>Pas de commentaire</span>

@endforelse

**-Relation one to one (1,1) voir dans post model**

public function image()

    {

        return $this->hasOne(image::class);

    }

\*recuperation: {{ $post->image->path }}//path est une fonstion dans post

-**Relation many to many (table association)**

Exemeple:post\_tag//association table on doit respecter l’ordre alphabetique c entre post et tag

-Remarque :créer une migration :

\*php artisan make:migration crea

te\_pivot\_table\_post\_tag

exemple de relation

 public function tags()

    {

        return $this->belongsToMany(tags::class);//dans le modéle post

    }

**Relation one to many (polymorphique)**

Exemple : Par exemple, imaginez que les utilisateurs de votre application puissent « commenter » des publications et des vidéos.

-Pour faire un enregistrement avec savemany et polymorphisme : //voir fonction register

 $post=Post::find(11);

$comment1=new comment(['content'=>'Mon premier commentaire']);

        $comment2=new comment(['content'=>'Mon deuxieme commentaire']);

        $post->comments()->saveMany([//comments est une fonction dans post

            $comment1,

            $comment2

        ]);

**RELEATION HAS ON THROUGHT(ONE TO ONE)**

-voir relation entre post\_image\_artist

Exemple : dans post :

 public function imageArtist ()

    {

        return $this->hasOneThrough(Artist::class, image::class);

    }

-remarque c’est dans la table qui donne sa clé on fait :

 return $this->hasOne(image::class);

**RELEATION** has-one-of-many

Resumé Parfois, un modèle peut avoir de nombreux modèles connexes, mais vous souhaitez facilement récupérer le modèle lié « le plus récent » ou « le plus ancien » de la relation. Par exemple, un modèle peut être lié à de nombreux modèles, mais vous souhaitez définir un moyen pratique d’interagir avec la commande la plus récente que l’utilisateur a passée. Vous pouvez y parvenir à l’aide du type de relation combiné aux méthodes suivantes :UserOrderhasOneofMany

Le plus récent commentaire et le dernier commentaire :

    public function latestcomment()

{

    return $this->hasOne(comment::class)->latestOfMany();

}

public function oldestcomment()

{

    return $this->hasOne(comment::class)->oldestOfMany();

}

**Les requêtes**

****

**-RECUP2RER UN INPUT**

$request->input('title')

**-**voir le path

$request->path() ;

-pour vérification path :

$request->is('create')

-voir méthode :

$request->method() ;

-vérification méthode formulaire :

$request->isMethod(‘Post’) ;

-**Remarque :** prérequis fichier : enctype="multipart/form-data" //dans la balise form

Upload image :

       $request->avatar->store('avatar');//dans le dossier avatar et le nom avatar

**Validation formulaire:**

 $request->validate([

           'title'=>'required|unique:nom\_table',

           'content'=>''

      ]);//si la validation du formulaire est incorrect il redirige sur la méme page sans condition au préalable

Parmis les régles on a unique

-Message de validation :

On modifie config/app le variable local en fr puis dans ressources/lang on crée un dossier fr contenant les fichiers de lang/en

-créer une régle de validation

php artisan make:rule Uppercase

et dans APP/rule/uppercase

dans la fonction passe on fait notre régle

exemple d’utilisation :

 $request->validate([

           'title'=>['required','unique:posts','max:255','min:5',new Uppercase],

           'content'=>['required']

      ]);

**FICHIER**

-écrire dans un fichier :

   Storage::disk('local')->put('eample.txt','Mon contenu de text');//stocké dans storage/app

-recupération fichier :

$request->file('avatar')

Exemple d’utilisation :

   Storage::disk('local')->put('avatar',$request->file('avatar'));//mettre dans le dossier avatar et on peut aussi changer le chemin de stockage en mettant public auliu de local

Affichage image :

$content=Storage::get($name);

Vérifier si un image exists :

Storage::disk('local')->exists($name)

Télécharger un image:

Storage::download($name)

url du fichier:

Storage::url($name);

Le chemin du fichier :

Storage::path($name)

Enregistrer image :

Storage::put('avatartest',$request->avatar);

Ou

$request­->file(‘avatar’)->store(‘avatartest2’);

Accéssibilité image:

php artisan storage:link

affichage image:

<img src="{{ asset('storage/').'/'.$post->image->path }}" alt="image non dispo"/>

Authentification:

Dss on utilise un middleware pour empécher un accés sans connexion:

  public function \_\_construct(){

        $this->middleware('auth')->except('bar');

    }

-Pour cacher certaines informations dans la vue pour les non connecté :

On fait :

@auth

Contenu html

@endauth

-affichage nom de l’utilisateur connecté

{{ Auth::user()->name}}

-affichage pas connecté @guest contraire de @auth

@guest

html

@endguest

**Authorization : on utilise les gates**

-on ajoute d’abord une colonne parexemple admin qui contient les roles :

php artisan make:migration add\_admin\_column\_to\_users\_table --table=users

Dans la migration:

public function up()

    {

        Schema::table('users', function (Blueprint $table) {

            //

$table->boolean('admin')->default(0);//ligne ajouté

        });

    }

1)étapes 1 on vas dans provider/authsericeprovider

  public function boot()

    {

        $this->registerPolicies();

        //

        Gate::define('admin-access',function(User $user){//la ligne ajouté

            return $user->admin;

        });

    }

2)puis dans le controlleur:

  if(!Gate::allows('admin-access')){

        abort('403');

        }