Pour afficher la liste des commandes artisan

php artisan list

Pour afficher l’aide artisan

php artisan --help

Pour ajouer un fichier de configuration

php artisan config:publish

pour installer les composants node js contenus dans le fichier package.json

npm install

Pour démarrer le build en mode developpement

npm run dev ou bien composer run dev

Pour démarrer le build en mode production

npm run build

# Creation d’un mini blog

Initilisez votre repo git local

Git init  
git add .  
git commit -m ‘initialisation’

Créer le fichier de migration de la table posts

php artisan make:migration create\_posts\_table

modifier le contenu du fichier du migration

public function up(): void

{

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

$table->id();

$table->string('title');

$table->text('content');

$table->timestamps(); // created\_at et updated\_at

});

}

Lancer la migration (le système ne va migrer que les nouveaux fichiers)

php artisan migrate

Lancer la migration (le système va effacer la base de données et refaire toute la migration a novueau)

php artisan migrate:fresh

Créer le modèle Post

php artisan make:model Post

Modifier son contenu

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Factories\HasFactory;

class Post extends Model

{

/\*\* @use HasFactory<\Database\Factories\UserFactory> \*/

use HasFactory;

/\*\*

\* The attributes that are mass assignable.

\*

\* @var list<string>

\*/

protected $fillable = [

'title',

'content',

];

}

Créer le factory

php artisan make:factory PostFactory

public function definition(): array

{

return [

'title' => fake()->sentence(),

'content' => fake()->paragraphs(3, true),

];

}

ajouter un seeder (s’il n’existe pas)

php artisan make:seeder DatabaseSeeder

ajouter cette ligne après la creation du l’utilisateur déjà existant

….   
Post::factory(10)->create();

….

Lancer le remplissage de la base de données avec des données de démo

php artisan db:seed

Les routes

Créer le controller PostController

php artisan make:controller PostController

Ajouter ces routes dans le fichier routes/web.php

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

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

Ajouter ces deux procedures au fichier PostController

public function index()

{

$posts = Post::all();

return view('posts.index', compact('posts'));

}

public function show(Post $post)

{

return view('posts.show', compact('post'));

}

Ajouter le dossier **posts** dans resouces/view

Ajouter les fichiers **index.blade.php** et **show.blade.php** dans le dossier **posts**

Remplissez les deux fichiers avec le code suivant :  
index.blade.php

<h1 >Articles du blog</h1>

@foreach ($posts as $post)

<div >

    <div >

        <h2 >{{ $post->title }}</h2>

        <p >{{ Str::limit($post->content, 100) }}</p>

        <a href="/posts/{{ $post->id }}" >Lire la suite</a>

    </div>

</div>

@endforeach

Show.blade.php

<div >

    <div >

        <h1 >{{ $post->title }}</h1>

        <p >{{ $post->content }}</p>

    </div>

</div>

Testez l’application avec (si c’est pas en cours d’execution)

php artisan serve

Ajout des commentaires

-----------------------

Créer en même temps un modèle, une migration, un contoller et un factory

php artisan make:model Comment -mcf

dans la migration des comments

Définir la table comments

Schema::create('comments', function (Blueprint $table) {

$table->id();

$table->text('content');

$table->foreignId('post\_id')->constrained()->onDelete('cascade');

$table->timestamps();

});

Oubien  
  
Schema::create('comments', function (Blueprint $table) {

$table->id();

$table->text('content');

$table->unsignedBigInteger('post\_id')->constrained();

$table->foreign('post\_id')->references('id')->on('posts')->onDelete('cascade')->onUpdate("cascade");

$table->timestamps();

});

dans le modèle Post (définir la relation avec comment)

public function comments()

{

return $this->hasMany(Comment::class);

}

dans le modèle Comment (définir les champs et la reation avec post)

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

use App\Models\Post;

class Comment extends Model

{

/\*\* @use HasFactory<\Database\Factories\CommentFactory> \*/

use HasFactory;

/\*\*

\* The attributes that are mass assignable.

\*

\* @var list<string>

\*/

protected $fillable = [

'content',

'post\_id'

];

public function post()

{

return $this->belongsTo(Post::class);

}

}

dans le factory comment (définir le remplissage des données demo)

public function definition(): array

{

return [

'content' => fake()->paragraph(),

'post\_id' => Post::inRandomOrder()->first()->id,

];

}

dans DatabaseSeeder (ajouter 3 commentaire pour chaque post)

$posts = Post::factory(10)->create();

$posts->each(function ($post) {

Comment::factory(3)->create(['post\_id' => $post->id]);

});

refaire la migration (fresh pour reprendre a zero et seed pour remplir la base de données)

php artisan migrate:fresh

php artisan db:seed  
  
ou bien  
  
php artisan migrate:fresh --seed

regardez aussi  
php artisan migrate:rollback --setp=1  
et  
php artisan migrate:rollback

ajouter la route dans routes/web.php

Route::post('/posts/{post}/comments', [CommentController::class, 'store']);

Ajouter les commentaire et le formulaire d’ajout dans la fiche du post

modifier le code de posts/show.blade.php par

<h1>{{ $post->title }}</h1>

<p>{{ $post->content }}</p>

<h2>Commentaires ({{ $post->comments->count() }})</h2>

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

<div style="margin-left: 20px; border-left: 2px solid #ccc; padding-left: 10px;">

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

<small>{{ $comment->created\_at->diffForHumans() }}</small>

</div>

@endforeach

<!-- Formulaire d'ajout de commentaire -->

<form method="POST" action="/posts/{{ $post->id }}/comments">

@csrf

<textarea name="content" placeholder="Votre commentaire..." required></textarea>

<button type="submit">Ajouter</button>

</form>

Ajoutez la methode store au CommentController

 public function store(Post $post)

    {

        request()->validate(['content' => 'required|min:3']);

        $post->comments()->create([

            'content' => request('content')

        ]);

        return back(); // Redirige vers la page du post

    }

Testez l’application avec (si c’est pas en cours d’execution)

php artisan serve

Ajotuer une layout pour la logique de prénsetation et du design

créer le controller BlogController et ajouter le code suivant

php artisan make:controller BlogController

ajouter cette fonction dans le controller BlogController

public function about()

{

return view('about');

}

ajouter la route dans routes/web.php

Route::get('/about', [BlogController::class, 'about']);

créer le fichier layouts/app.blade.php

<!DOCTYPE html>

<html lang="fr">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>ISMO - Mon Blog - @yield('title')</title>

<!-- Intégration de Bootstrap (exemple) -->

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">

</head>

<body>

<!-- Navigation -->

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<div class="container">

<a class="navbar-brand" href="/">Mon Mini Blog</a>

<div class="navbar-nav">

<a class="nav-link {{ Request::is('/') ? 'active' : '' }}" href="/">Articles</a>

<a class="nav-link {{ Request::is('about') ? 'active' : '' }}" href="/about">À propos</a>

</div>

</div>

</nav>

<!-- Contenu principal -->

<div class="container mt-4">

@yield('content')

</div>

<!-- Footer -->

<footer class="mt-5 bg-light py-3 text-center">

<p>© 2025 Mon Blog Laravel. Tous droits réservés.</p>

</footer>

</body>

</html>

Modifier about.blade.php

@extends('layouts.app')

@section('title', 'A propos de')

@section('content')

    <h1>

        mini blog 2025 - Tous les droits sont réservés.

    </h1>

@endsection

modifier posts/index.blade.php

@extends('layouts.app')

@section('title', 'Liste des articles')

@section('content')

<h1 class="mb-4">Articles du blog</h1>

@foreach ($posts as $post)

<div class="card mb-3">

<div class="card-body">

<h2 class="card-title">{{ $post->title }}</h2>

<p class="card-text">{{ Str::limit($post->content, 100) }}</p>

<a href="/posts/{{ $post->id }}" class="btn btn-primary">Lire la suite</a>

</div>

</div>

@endforeach

@endsection

modifier posts/show.blade.php

@extends('layouts.app')

@section('title', $post->title)

@section('content')

<div class="card">

<div class="card-body">

<h1 class="card-title">{{ $post->title }}</h1>

<p class="card-text">{{ $post->content }}</p>

</div>

</div>

<div class="mt-4">

<h3>Commentaires ({{ $post->comments->count() }})</h3>

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

<div class="card mb-2">

<div class="card-body">

<p class="card-text">{{ $comment->content }}</p>

<small class="text-muted">

Posté {{ $comment->created\_at->diffForHumans() }}

</small>

</div>

</div>

@endforeach

<form method="POST" action="/posts/{{ $post->id }}/comments" class="mt-4">

@csrf

<div class="mb-3">

<textarea name="content" class="form-control" placeholder="Votre commentaire..." required></textarea>

</div>

<button type="submit" class="btn btn-success">Ajouter un commentaire</button>

</form>

</div>

@endsection

Testez l’application avec (si c’est pas en cours d’execution)

php artisan serve