



Laravel + Vue = ❤️

Alexandre TOBIA - 08/03/21



# Sommaire

Présentation du cours et ses objectifs

Inertia ?

Installation de Jetstream + Inertia

Installation Chrome DevTools

Présentation structure Inertia

Utilisation d'Inertia



# Présentation du cours et ses objectifs

Ce cours aura pour objectif de vous montrer comment utiliser ensemble les deux dernières technologies que vous avez utilisé, à savoir, Laravel et Vue JS



# Les différentes façons de le faire

Plusieurs méthodes :

- Le front serait réalisé via un framework (nuxt par exemple) et le back via Laravel avec une communication via API.  
Totalement réalisable et fonctionnelle mais peut-être compliqué à mettre en place (cors, nécessité de mettre en place un système pour autoriser les requêtes, maintenance, plusieurs repositories,...)
- La deuxième, serait de totalement lier les deux via une approche plus inclusive, c'est celle-ci que nous verrons dans ce cours.

# Inertia

**inertia**» THE MODERN MONOLITH

Q Search...



## Build single-page apps, without building an API.

Inertia.js lets you **quickly build modern single-page React, Vue and Svelte apps** using classic server-side routing and controllers.

UsersController.php

Users.vue

```
class UsersController
{
    public function index()
    {
        $users = User::active()
            ->orderBy('name')
            ->get()
            ->only('id', 'name', 'email');

        return Inertia::render('Users', [
            'users' => $users
        ]);
    }
}
```



# Inertia

Inertia vous permet de créer des SPA entièrement rendues côté client, sans la complexité des SPA modernes. Pour ce faire, elle s'appuie sur la façon de faire côté back-end.

Ici pas de router en Vue, on garde le router de laravel, inertia s'occupe d'afficher les bonnes pages/composant en fonction de l'url.



# Installation de chrome dev tools pour vue

<https://chrome.google.com/webstore/detail/vuejs-devtools/ljjemllljcmogpfapbkkighbhppjdbg/related?hl=en>



# Installation de laravel

En ligne de commande via l'outil "laravel" ou via composer :



```
laravel new inertia-laravel
```

```
// OR
```

```
composer create-project laravel/laravel inertia-laravel
```

Le nom du dossier aura pour nom "inertia-laravel" dans mon exemple. Pensez à bien mettre ce dossier dans le répertoire utilisé par votre gestionnaire de site. (Exemple : Laragon, Wamp,...)





# Configuration

Créez une base de données via  
phpmyadmin.

Ensuite, ouvrez votre projet avec votre  
IDE. Il faut modifier le fichier `.env` avec  
vos informations de connexion mysql,  
et y mettre le nom de votre base de  
données.

Si le fichier `.env` n'existe pas, dupliquez  
le `.env.example` en `.env`.

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:FW4bcTZCNa0GDyrxC4EF9J0Gjrn3qI3028zMNI40jiI=
APP_DEBUG=true
APP_URL=http://localhost

LOG_CHANNEL=stack
LOG_LEVEL=debug

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=first_api
DB_USERNAME=root
DB_PASSWORD=
...
```



# Installation de Jetstream + Inertia

Pour utiliser Inertia et avoir une base de notre application (connexion/inscription) fonctionnelle, nous allons utiliser Jetstream avec la stack Inertia :




```
composer require laravel/jetstream  
  
php artisan jetstream:install inertia
```



# Installation de Jetstream + Inertia

Installation des différentes dépendances javascript, lançons les migrations puis notre watcher :

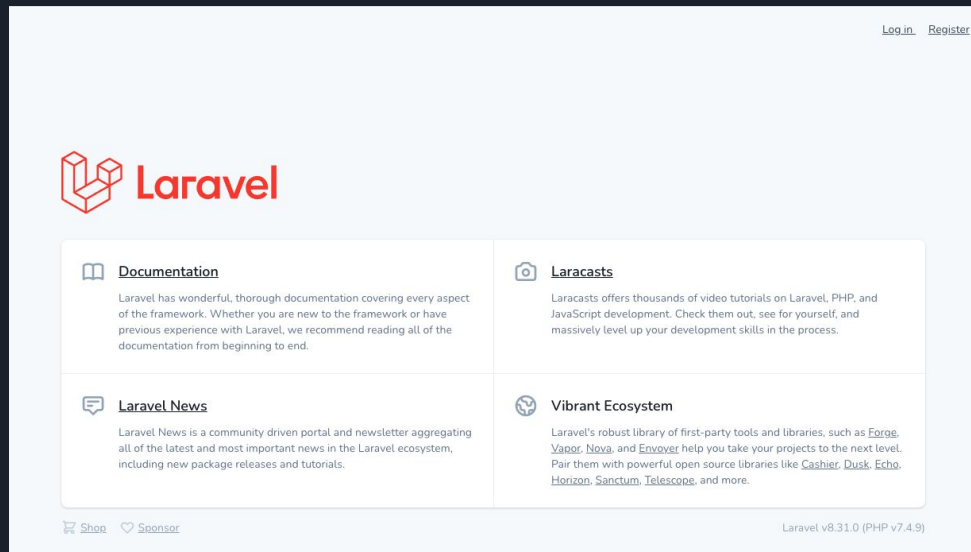


```
npm install  
php artisan migrate  
npm run hot
```

Ici nous utilisons “npm run hot” qui permet de lancer le “hot reloader” (rechargement à chaud) ce qui permet de recharger uniquement la partie modifiée (et non la page entière)

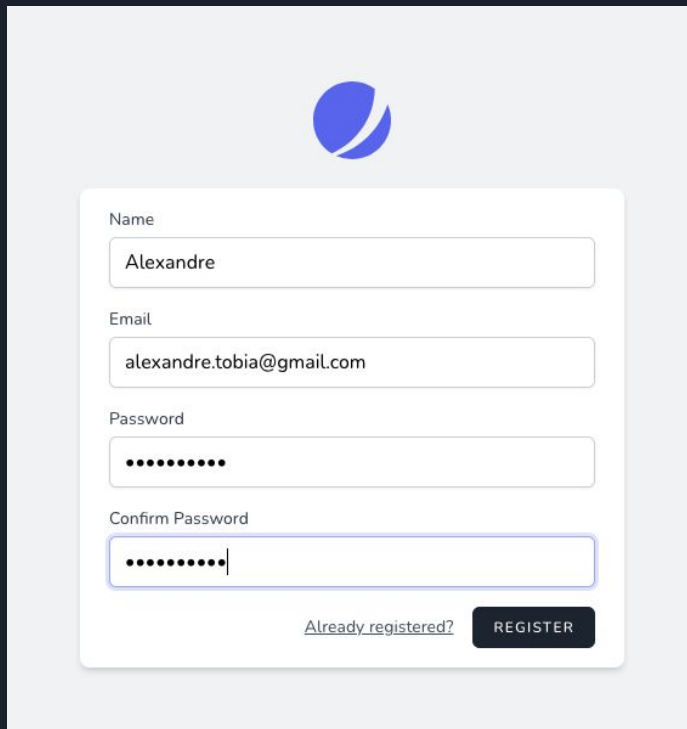
# Installation de Jetstream + Inertia

Vous devriez avoir ceci en vous rendant sur votre site en local




# Installation de Jetstream + Inertia

En cliquant sur “Register” vous pouvez vous créer un compte.



The screenshot shows a registration form with a blue circular logo at the top. The form contains four input fields: Name (filled with 'Alexandre'), Email (filled with 'alexandre.tobia@gmail.com'), Password (filled with dots), and Confirm Password (filled with dots). At the bottom right, there is a link for '[Already registered?](#)' and a dark blue 'REGISTER' button.



Name

Alexandre

Email

alexandre.tobia@gmail.com

Password

.....

Confirm Password

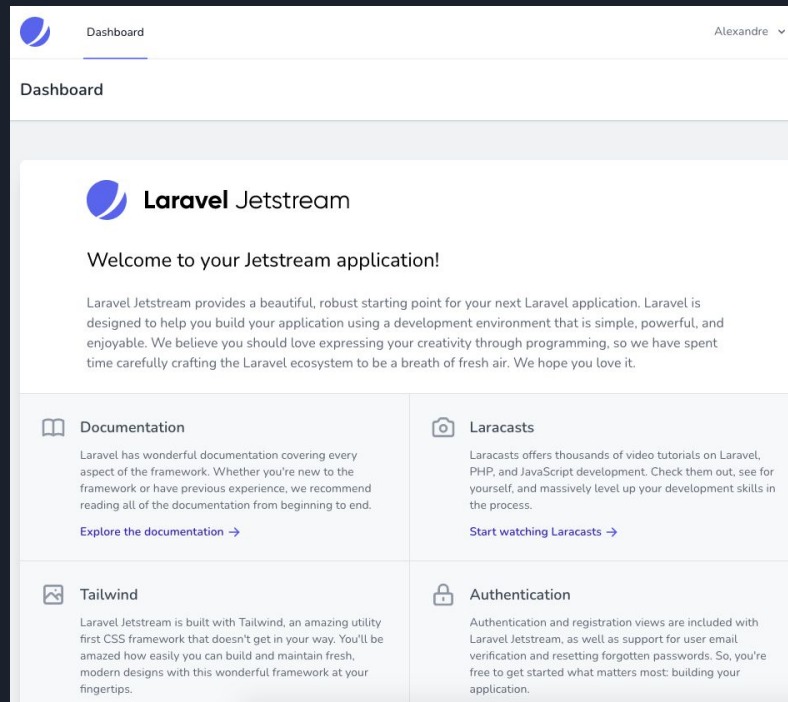
.....

[Already registered?](#)

REGISTER

# Installation de Jetstream + Inertia

Vous arrivez ensuite sur votre dashboard





# Comment fonctionne Inertia ?

Vous vous dites surement “rien de bien différent que la semaine sur Laravel, nous savons déjà utiliser Jetstream...”, vous n’avez pas totalement tort, à une exception qui est que l’ensemble de votre interface front est faites en Vue !

# Comment fonctionne Inertia ?

Notre point d'entrée reste un fichier blade, légèrement modifié (resources/views/app.blade.php).

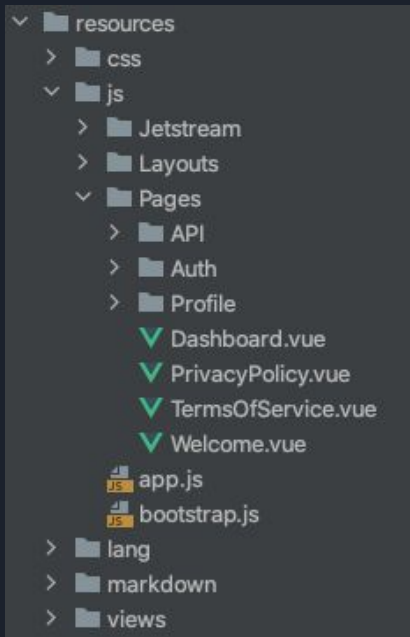
Ligne 21, @inertia permet d'ajouter automatiquement l'ensemble du code nécessaire à son bon fonctionnement, il ne faut surtout pas le supprimer !

```
1 <!DOCTYPE html>
2 <html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width, initial-
scale=1">
6     <meta name="csrf-token" content="{{ csrf_token() }}">
7
8     <title>{{ config('app.name', 'Laravel') }}</title>
9
10    <!-- Fonts -->
11    <link rel="stylesheet" href="https://fonts.googleapis.com/css2?
family=Nunito:wght@400;600;700&display=swap">
12
13    <!-- Styles -->
14    <link rel="stylesheet" href="{{ mix('css/app.css') }}">
15
16    <!-- Scripts -->
17    @routes
18    <script src="{{ mix('js/app.js') }}" defer></script>
19  </head>
20  <body class="font-sans antialiased">
21    @inertia
22  </body>
23 </html>
```



# Comment fonctionne Inertia ?

Notre page “Dashboard” est en réalité un fichier .vue



```
1 <template>
2   <app-layout>
3     <template #header>
4       <h2 class="font-semibold text-xl text-gray-800 leading-tight">
5         Dashboard
6       </h2>
7     </template>
8
9     <div class="py-12">
10      <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
11        <div class="bg-white overflow-hidden shadow-xl sm:rounded-lg">
12          <welcome></welcome>
13        </div>
14      </div>
15    </div>
16  </app-layout>
17 </template>
18
19 <script>
20   import AppLayout from '@Layouts/AppLayout'
21   import Welcome from '@Jetstream/Welcome'
22
23   export default {
24     components: {
25       AppLayout,
26       Welcome,
27     },
28   },
29 </script>
```

# Comment fonctionne Inertia ?

Pour qu'Inertia puisse afficher cette page, rendons-nous dans le fichier web.php, qui nous sert de router:

```
1 Route::get('/', function () {
2     return Inertia::render('Welcome', [
3         'canLogin' => Route::has('login'),
4         'canRegister' => Route::has('register'),
5         'laravelVersion' => Application::VERSION,
6         'phpVersion' => PHP_VERSION,
7     ]);
8 });
9
10 Route::middleware(['auth:sanctum', 'verified'])->get('/dashboard', function () {
11     return Inertia::render('Dashboard');
12 }->name('dashboard');
```



# Comment fonctionne Inertia ?

Un middleware “auth:sanctum” et “verified” nous permet d’éviter que n’importe qui puisse accéder à cette page (il faut être connecté).

À la place d’un classique “return view(“Dashboard”, nous avons “Inertia::render(“Dashboard”)” qui permet de dire: “Quand nous arrivons sur l’url “/dashboard” je veux que tu charges le composant “**Dashboard.vue**” (se trouvant à la racine de notre dossier “**Pages**” dans notre dossier “js”. Si notre fichier “Dashboard.vue” se trouvait dans un sous-dossier “Admin”, il aurait fallu écrire “**return Inertia::render(‘Admin/Dashboard’);**”

```
9
10 Route::middleware(['auth:sanctum', 'verified'])->get('/dashboard', function () {
11     return Inertia::render('Dashboard');
12 }->name('dashboard');
```



# Comment fonctionne Inertia ?

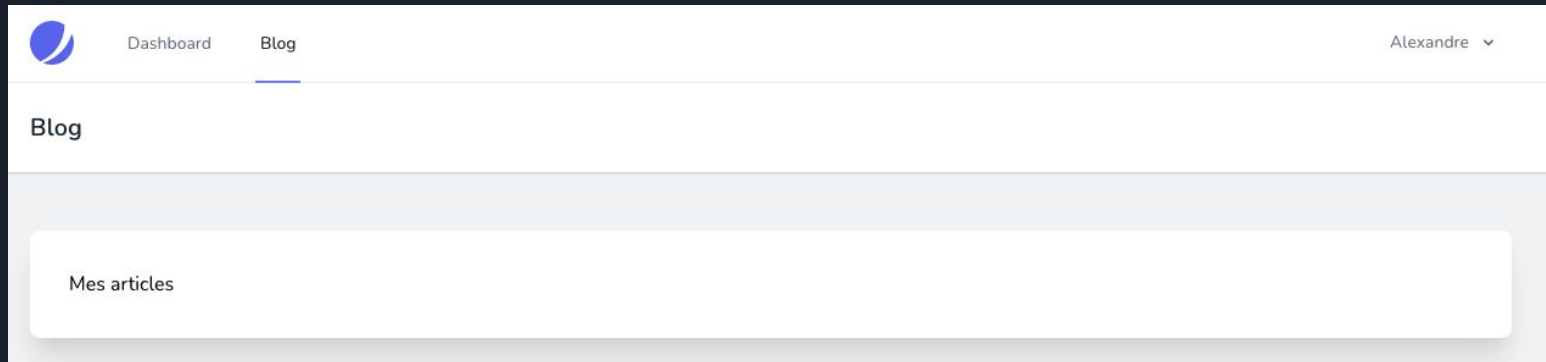
Dans notre fichier “Dashboard.vue”, l’ensemble de notre page est englobée dans un composant “app-layout” qui n’est autre que le layout de notre page (se trouvant dans `resources/js/Layouts/AppLayout.vue`)

# Premier exercice :

En respectant la logique que je vous ai présentée

1. Créer un controller “BlogController” avec une fonction “index”
2. Créer une route “/blog” relié à notre fonction qui affichera une page en .vue
3. Modifier le menu global pour y ajouter “Blog” (à côté de “Dashboard”)

Résultat:



# Envoyer des données à nos vues

Pour envoyer des données à nos vues (props) il faut passer en second argument, un tableau à la fonction “render” d’Inertia.

Désormais notre composant “Index” va pouvoir recevoir une props du nom de “posts”

```
1 <script>
2 import AppLayout from '@/Layouts/AppLayout'
3
4 export default {
5   props: ['posts'],
6   components: {
7     AppLayout,
8   },
9 }
10 </script>
```

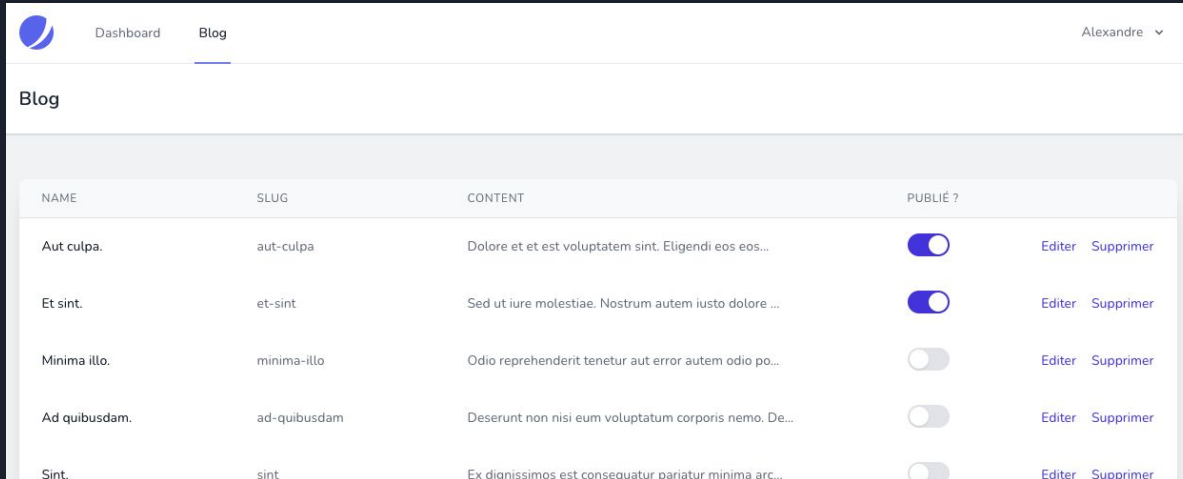
js/Pages/Blog/Index.vue

```
1 public function index()
2 {
3     return Inertia::render('Blog/Index', [
4         'posts' => [
5             [
6                 "name" => "Mon article",
7                 "slug" => 'mon-article',
8                 'content' => 'lorem ipsum',
9                 'published' => 0
10            ],
11            ["name" => "Mon deuxieme article",
12             "slug" => 'mon-deuxieme-article',
13             'content' => 'lorem ipsum',
14             'published' => 1]
15            //...
16        ]
17    ]);
18 }
```

app/Http/Controllers/BlogController.php

# Deuxième exercice :

1. Créer un model "Post" avec sa migration
  - a. (string) name
  - b. (string) slug (unique)
  - c. (text) content
  - d. (boolean) published
2. Créer une factory pour créer 50 posts différents via un seeder
3. Afficher l'ensemble des posts sous cette forme (code html du tableau: <https://bit.ly/30mNhNy>) :



The screenshot shows a web application interface with a top navigation bar containing a logo, 'Dashboard', 'Blog' (active), and a user profile 'Alexandre'. Below the navigation bar, the 'Blog' section is titled. A table displays a list of blog posts with columns for NAME, SLUG, CONTENT, and PUBLIÉ ?. Each row includes a toggle switch for the 'PUBLIÉ ?' status and links for 'Editer' and 'Supprimer'.

NAME	SLUG	CONTENT	PUBLIÉ ?	
Aut culpa.	aut-culpa	Dolore et et est voluptatem sint. Eligendi eos eos...	<input checked="" type="checkbox"/>	Editer Supprimer
Et sint.	et-sint	Sed ut iure molestiae. Nostrum autem iusto dolore ...	<input checked="" type="checkbox"/>	Editer Supprimer
Minima illo.	minima-illo	Odio reprehenderit tenetur aut error autem odio po...	<input type="checkbox"/>	Editer Supprimer
Ad quibusdam.	ad-quibusdam	Deserunt non nisi eum voluptatum corporis nemo. De...	<input type="checkbox"/>	Editer Supprimer
Sint.	sint	Ex dionissimos est consequat pariatur minima arc...	<input type="checkbox"/>	Editer Supprimer

# Envoyer une requête post, put, patch avec Inertia

Pour envoyer des données (ajax) nous avons accès à l'objet "this.\$inertia" pour y parvenir.

```
1 public function maFonction($id)
2 {
3     //Code de ma fonction
4     return redirect()->back();
5 }
```

Vous devrez toujours renvoyer une réponse "redirect" pour dire à Inertia quoi faire (dans cet exemple, on renvoie sur la même page ce qui équivaut à un reload de la page)

```
<template>
  <button @click="maFonction">Mon bouton</button>
</template>

<script>
export default {
  methods: {
    maFonction() {
      const params = {
        param1: "value",
        param2: "value2",
        //...
      }
      this.$inertia.post('/url-cible', params)
    }
  }
}
</script>
```

Vous avez aussi accès à une fonction "this.route()", qui vous permet d'appeler vos routes nommé dans le web.php. Exemple: à la place de "/url-cible" on aurait pu avoir :  
this.route('nom-de-ma-route', post->id)



## Troisième exercice :

1. Faire en sorte de pouvoir “Publié” / “Dépublier” un article en cliquant sur le “toggle”
  - a. Il faut que le champ “published” passe de 0 à 1 (et inversement) en base de donnée.

NAME	SLUG	CONTENT	PUBLIÉ ?	
Aut culpa.	aut-culpa	Dolore et et est voluptatem sint. Eligendi eos eos...	<input checked="" type="checkbox"/>	<a href="#">Editer</a> <a href="#">Supprimer</a>
Et sint.	et-sint	Sed ut iure molestiae. Nostrum autem iusto dolore ...	<input type="checkbox"/>	<a href="#">Editer</a> <a href="#">Supprimer</a>
Minima illo.	minima-illo	Odio reprehenderit tenetur aut error autem odio po...	<input checked="" type="checkbox"/>	<a href="#">Editer</a> <a href="#">Supprimer</a>

# La gestion des formulaires

Vous devez renseigner un objet “form” via “this.\$inertia.form”. On dit ici que lors du submit, on appelle notre methods “submit”.

```
export default {
  data() {
    return {
      form: this.$inertia.form({
        email: null,
        password: null,
        remember: false,
      }),
    },
    methods: {
      submit() {
        this.$inertia.post('/url', this.form)
      }
    }
  }
}
```

```
<form @submit.prevent="submit">
  <!-- email -->
  <input type="text" v-model="form.email">
  <div v-if="form.errors.email">{{ form.errors.email }}</div>
  <!-- password -->
  <input type="password" v-model="form.password">
  <div v-if="form.errors.password">{{ form.errors.password }}</div>
  <!-- remember me -->
  <input type="checkbox" v-model="form.remember"> Remember Me
  <!-- submit -->
  <button type="submit" :disabled="form.processing">Login</button>
</form>
```



# La gestion des formulaires

On valide la requête, puis nous effectuons une redirection.

```
public function store()
{
    Request::validate([
        'first_name' => ['required', 'max:50'],
        'last_name' => ['required', 'max:50'],
        'email' => ['required', 'max:50', 'email'],
    ]);

    $user = User::create(
        Request::only('first_name', 'last_name', 'email')
    );

    return Redirect::route('users.show', $user);
}
```



# La gestion des formulaires

On valide la requête, puis nous effectuons une redirection.

```
public function store()
{
    Request::validate([
        'first_name' => ['required', 'max:50'],
        'last_name' => ['required', 'max:50'],
        'email' => ['required', 'max:50', 'email'],
    ]);

    $user = User::create(
        Request::only('first_name', 'last_name', 'email')
    );

    return Redirect::route('users.show', $user);
}
```

## Quatrième exercice :

1. Créer une route “blog/create” affichant un formulaire pour créer un article
2. Faire le code PHP pour enregistrer l'article.
3. Afficher les messages d'erreurs (errorBag)
4. Faites de même pour la route d'edition (blog/{id}/edit

Nom

The name field is required.

Slug

The slug field is required.

Contenu

The content field is required.

Publié ?

☐

SAVE

# TD

1. Créer une route “blog/create” affichant un formulaire pour créer un article
2. Faire le code PHP pour enregistrer l'article.
3. Afficher les messages d'erreurs (errorBag)
4. Faites de même pour la route d'edition (blog/{id}/edit

Nom

The name field is required.

Slug

The slug field is required.

Contenu

The content field is required.

Publié ?

☐

SAVE



# TD

## Consignes :

- Par groupe de deux, avec Laravel, & Inertia vous devrez développer ce dashboard:  
<https://cutt.ly/Wzxc6nA>
- Avant de vous lancer tête baissée dans le rendu plusieurs étapes à réaliser :
  - Écrire le cahier des charges du projet. C'est à dire, comment le site va fonctionner, les fonctionnalités attendues (par rapport à ce que vous voyez sur les maquettes) => rendu sur google doc
  - Faire le schéma de la base de données exemple: <https://cutt.ly/Bzxbg2K> afin d'illustrer les différents champs et relation entre les entités.
  - Me faire valider l'ensemble avant de commencer à développer
  - Choix du framework CSS, ici vous êtes libre d'utiliser Bootstrap si besoin (le framework par défaut est tailwind sur inertia)
  - Prévoir factory et seeders pour remplir la base de données
  - Rendu sur github Classroom: <https://classroom.github.com/g/Mn9Ctt93>