



Tutoriel création de site web Laravel X React

Abderrahman ALLOUH – Benjamin BOCK

Crédit : Antoine MOUCHAMPS

30 septembre 2025 - 30 septembre 2025



N-HiTec

Allée de la Découverte 10, 4000 Liège, Belgium
nhitec.com | info@nhitec.com

Table des matières



I. Introduction

Nous revoilà ! Si vous lisez ceci, j'espère que vous avez bien effectué la formation précédente sur l'installation de Docker  et Laravel 

ATTENTION ! Vous allez rapidement remarquer que de nombreuses notes de bas de page¹ sont présentes tout au long de la formation. Celles-ci contiennent souvent des informations importantes, donc il ne faut pas les passer. Lisez-les attentivement !

1 Laravel, c'est quoi ?

1.1 Prélude

Laravel  est ce qu'on appelle un *framework*. C'est à dire un ensemble d'outil fournissant une architecture de base sur laquelle n'importe quel site web peut être bâti. A la fin de ce "petit" tutoriel, vous serez je l'espère capable d'utiliser les fonctionnalités principales de Laravel, ainsi que les langages utilisés par ce framework et par la création de site web en général : PHP , HTML  et en allant un petit peu plus loin, CSS , JQuery 

Bon alors, et ce framework alors ? Comment fonctionne-t'il ?

1.2 Fonctionnement & philosophie

Laravel  utilise une architecture dite "MVC" (Modèle, Vue, Contrôleur) qui est décrite par la figure FIGURE ???. Elle se base donc sur 4 concepts :

1. **Routing**: Le routing est l'étape consistant à lier une URL, une route, à une action spécifique, qui sera effectuée par une méthode (dans le sens *Object-oriented-programming* du terme) contenue dans un controller.
2. **Controller**: Les controllers sont donc appellés par les routes, ce sont eux qui vont s'occuper de manipuler les données, effectuer x-y-z tâches, et enfin d'envoyer une certaine view à l'utilisateur.
3. **View**: Le concept de view est plutôt simple : Avec Laravel, chaque view correspond grossièrement à une page que l'utilisateur voit affichée sur son écran.
4. **Model**: Enfin, les données stockées dans la base de donnée ne sont pas traitées telles quelles. Laravel nous facilite la vie en associant chaque type de donnée à un model, qui sera plus simple à utiliser par les controllers et comportera des fonctionnalités très utiles.

2 React, c'est quoi ?

2.1 Prélude

React  est ce qu'on appelle une *bibliothèque Javascript* . Contrairement à un framework comme Laravel , React ne cherche pas à tout contrôler : son rôle principal, c'est de vous aider à créer des

1. Comme celle-ci



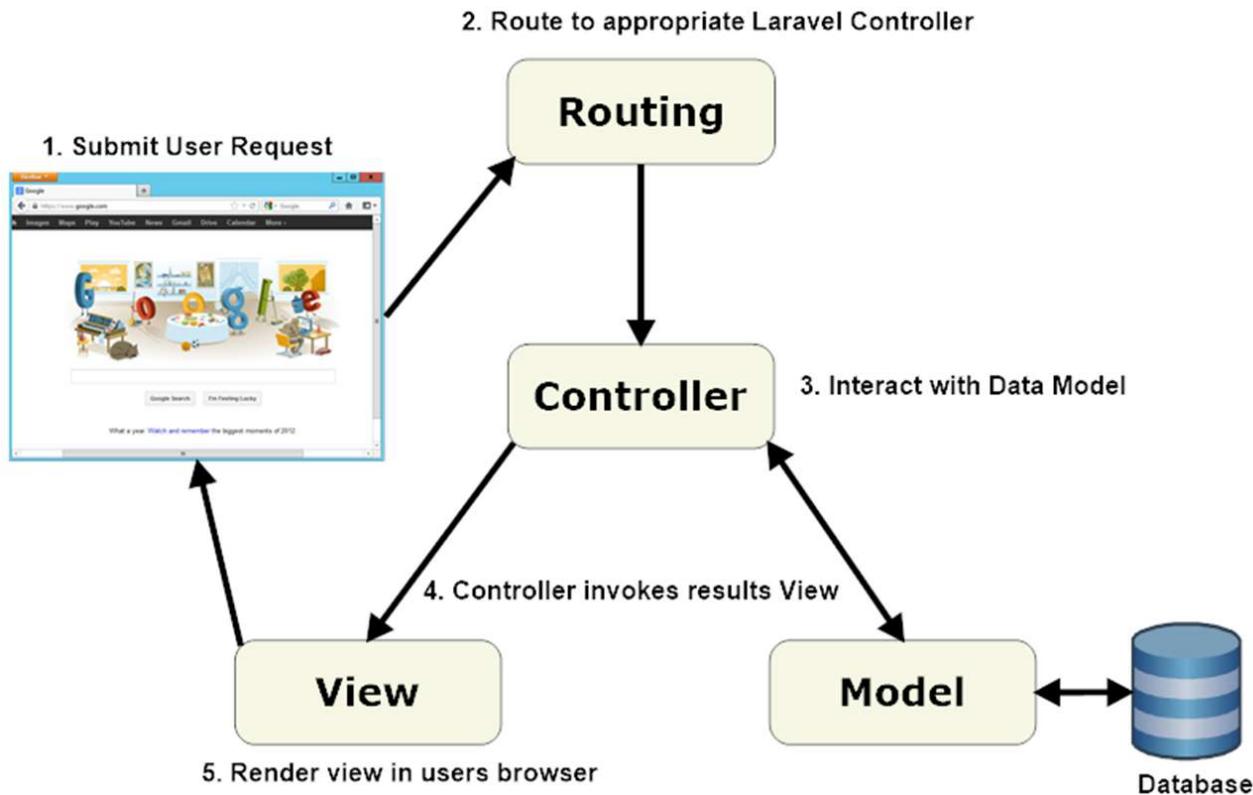


FIGURE 1

interfaces utilisateur (les pages web que vous voyez à l'écran).

Dit autrement : si Laravel 🚗 s'occupe des coulisses (*backend*), React, lui, gère ce que l'utilisateur voit et avec quoi il interagit (*frontend*).²

Astuce : Pensez à React 💡 comme à une boîte de Lego : chaque brique (un composant) peut être réutilisée pour construire des interfaces plus grandes et plus complexes.

2.2 Fonctionnement & philosophie

React repose sur quelques concepts clés :

1. Composants: Chaque partie de votre site (bouton, barre de navigation, formulaire, etc.) est un composant. Ces composants peuvent être combinés pour former une page complète.
 2. JSX: React 💡 utilise une syntaxe qui mélange Javascript💡 et HTML📝. Cela peut sembler étrange au début, mais c'est ce qui rend React 💡 puissant et lisible.
 3. Props: Les composants peuvent recevoir des informations de l'extérieur (comme des paramètres dans une fonction). On appelle ça des props.
-
2. Vous pouvez consulter la documentation officielle de React à l'adresse suivante : <https://react.dev>

4. État (State) : Un composant peut garder en mémoire des informations qui changent au fil du temps (par exemple : le contenu d'un champ de recherche). C'est ce qu'on appelle l'état.
5. Virtual DOM: React  ne modifie pas directement la page, mais une copie virtuelle de celle-ci. Quand quelque chose change, il met à jour uniquement la partie nécessaire, ce qui rend l'application rapide.

2.3 Pourquoi l'utiliser ?

Alors pourquoi se compliquer la vie avec React  ?

- Parce qu'il permet de créer des interfaces modernes, interactives et rapides.
- Parce qu'il est utilisé partout (Facebook, Instagram, Airbnb, Netflix...).
- Parce qu'il est **composantiel** : vous écrivez une fois un bouton, et vous pouvez le réutiliser 20 fois dans l'application sans copier-coller.
- Parce qu'il a une **énorme communauté** : dès que vous avez un problème, il y a probablement déjà une solution sur Internet.

Bref, React, c'est un peu comme passer d'un vieux Nokia 3310 à un smartphone : ça ouvre un nouveau monde de possibilités.

Maintenant que nous avons Laravel  pour gérer le **backend** et React  pour construire le **frontend**, il reste une question cruciale : *comment faire pour que ces deux mondes communiquent efficacement entre eux ?* C'est précisément là qu'intervient Inertia  . Dans la prochaine section, nous allons décorner ensemble comment Inertia nous permet de marier React et Laravel sans avoir besoin de construire une API séparée.

3 Inertia.js, c'est quoi ?

3.1 Prélude

Vous avez maintenant vu ce qu'est Laravel  et ce qu'est React  . La grande question qui se pose, c'est : *comment faire communiquer ces deux mondes ?*

Traditionnellement, un **backend** comme Laravel  enverrait ses données à travers une *API*³. Le **frontend**, ici React , consommerait ensuite ces données pour les afficher. Mais construire une API complète⁴ peut rapidement devenir lourd et complexe, surtout quand le but est juste de rendre vos pages React.

Inertia  agit comme une *colle magique* entre Laravel  et React  : il vous permet de continuer à écrire vos controllers et vos routes Laravel comme vous l'avez toujours fait, mais au lieu de retourner une vue blade, vous retournez une vue React  directement.

3. Une API, ou interface de programmation d'application, est un ensemble de règles et de spécifications qui permettent à différents logiciels de communiquer et d'échanger des données

4. Une API complète nécessite souvent de créer beaucoup d'endpoints, gérer les statuts d'erreur, les authifications, et parfois même écrire de la documentation pour l'utiliser correctement.

En pratique, ça veut dire :

- vous gardez toute la logique côté Laravel (routes, contrôleurs, modèles, etc.),
- Inertia ➔ transmet les données⁵ à vos composants React,
- vos pages sont rendues par React, mais sans devoir passer par la mise en place compliquée d'une API.

En clair : Inertia supprime la barrière entre le backend et le frontend.

Si vous êtes intéressés par la documentation officielle d'Inertia.js, vous pouvez la lire ici⁶.

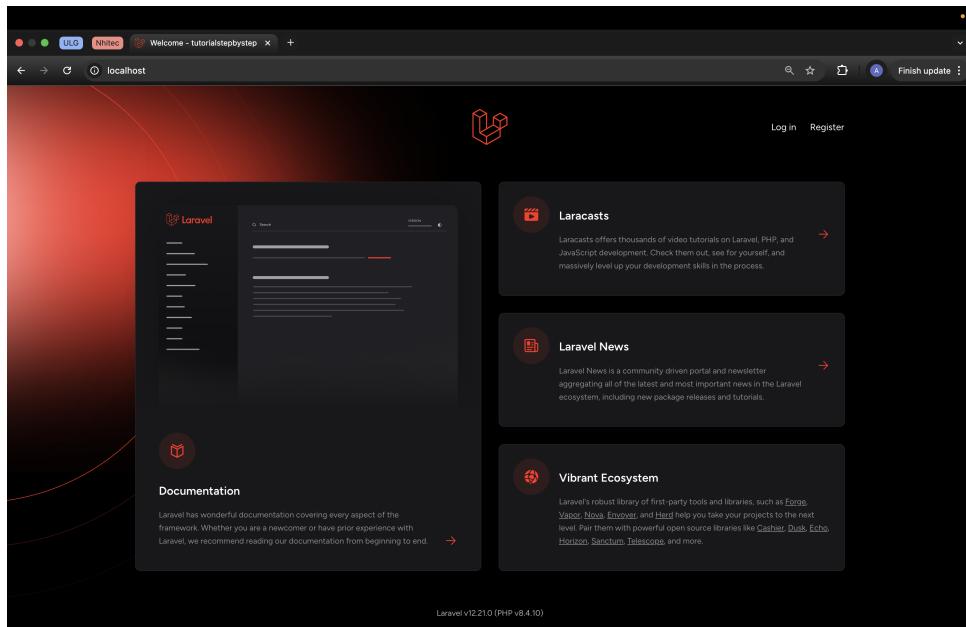
5. Sous le capot, Inertia utilise principalement le JSON pour transporter les données entre Laravel et React.
6. <https://inertiajs.com>



II. Premier site web

1 Setup initial

Toute cette partie est couverte par la formation précédente. Normalement, à la suite de tutoriel, vous devriez avoir obtenu le site suivant en vous rendant sur <http://lurlquevousavezchoisie> :⁷.



Nous allons partir de ce site là. Pour ce tutoriel, mon projet sera appellé `tutorialstepbystep` donc son URL sera <http://tutorialstepbystep/>.

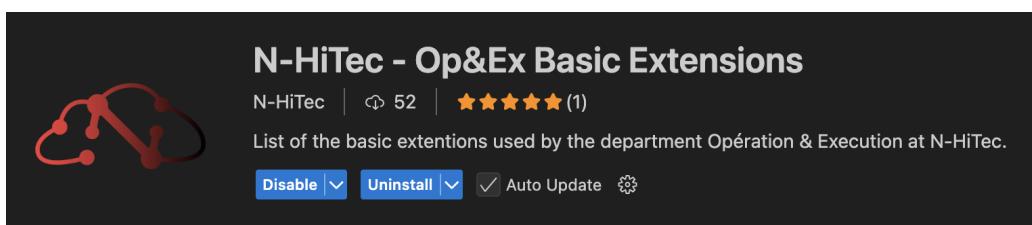
7. PS :n'oubliez pas de lancer `sail up -d` et `sail npm run dev` si cela n'est pas déjà fait ! Ne vous inquiétez pas, l'explication pour la deuxième commande va bientôt arriver...

1.1 Extensions

Vous allez rapidement remarquer de nouvelles fonctionnalités que mon IDE⁸ possède par rapport au vôtre. Ces fonctionnalités sont rajoutées grâce aux extensions de VS Code .

Pour éviter d'installer les extensions une par une, nous avons regroupé toutes les extensions utiles dans un pack unique.

- Ouvrez VS Code .
- Allez dans l'onglet Extensions dans la barre d'outils à gauche.
- Recherchez N-HiTec - Op&Ex Basic Extensions.
- Cliquez sur **Install**.



Grâce à ce pack, vous aurez directement un environnement de travail complet et prêt pour le développement.

2 Premières routes & composants React

2.1 welcome !

En Laravel  + Inertia , les routes se trouvent toujours dans routes/web.php. Une route, c'est simplement un chemin que Laravel surveille. Quand quelqu'un visite ce chemin dans le navigateur, Laravel sait quoi afficher.

Voici la route par défaut que vous trouverez dans un projet neuf :

```
Route::get('/', function () {
    return Inertia::render('Welcome', [
        'canLogin' => Route::has('login'),
        'canRegister' => Route::has('register'),
        'laravelVersion' => Application::VERSION,
        'phpVersion' => PHP_VERSION,
    ]);
});
```

8. Au cas où ça ne serait toujours pas clair, utilisez VS Code  !!

- Route::get('/', ...) signifie : « Quand quelqu'un va à l'adresse / (la page d'accueil), fais ce qu'il y a dans les accolades ».
- Ici, on utilise Inertia::render('Welcome') pour dire à Laravel : « Affiche la page React Welcome ».
- Welcome correspond à un fichier Welcome.jsx qui se trouve dans resources/js/Pages.

2.2 Welcome

Allons voir le fichier Welcome.jsx.

Par défaut, il contient beaucoup de code inutile pour notre formation : styles, textes et boutons de démonstration. On va tout effacer et repartir d'une base simple.

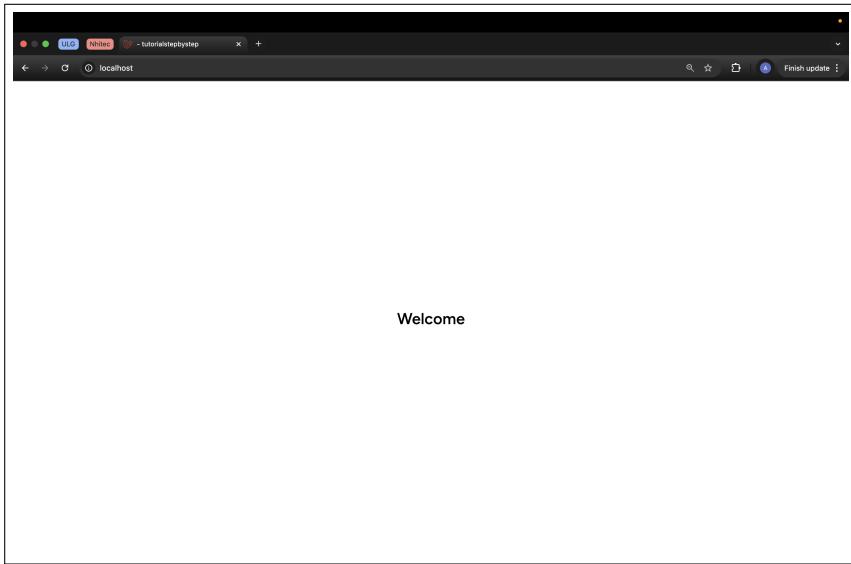
Voici à quoi ressemble le fichier Welcome.jsx après nettoyage. Il ne contient qu'un composant React minimal, prêt à accueillir notre contenu.

```
resources > js > Pages > JS Welcome.jsx > Welcome
1  export default function Welcome() {
2    return (
3      <div
4        style={{
5          display: "flex",
6          justifyContent: "center",
7          alignItems: "center",
8          height: "100vh",
9          fontSize: "2rem",
10         fontWeight: "bold",
11       }}
12     >
13       Welcome
14     </div>
15   );
16 }
17 }
```

Explication du code

- export default function Welcome() : on crée un composant React qui s'appelle **Welcome** et on l'exporte pour qu'il puisse être utilisé ailleurs.
- <div style={{ ... }}> : on crée une boîte (div) et on lui applique du style directement en JavaScript grâce à la syntaxe {{ ... }}.
- display: "flex" : active **Flexbox**, une méthode pratique pour centrer des éléments.
- justifyContent: "center" : centre le contenu horizontalement.
- alignItems: "center" : centre le contenu verticalement.
- height: "100vh" : la boîte prend toute la hauteur de l'écran (vh = *viewport height*).
- fontSize: "2rem" : définit la taille du texte à deux fois la taille normale.
- fontWeight: "bold" : met le texte en gras.





Résultat : une page toute blanche avec juste **Welcome** bien centré, qui servira de point de départ pour construire la suite du site.

À retenir :

1. Les routes définissent quel composant React sera affiché.
2. `Inertia::render('Welcome')` charge le fichier `Welcome.jsx`.
3. On part volontairement d'une page simple et vide pour mieux comprendre ce que l'on ajoute ensuite.

2.3 Controller

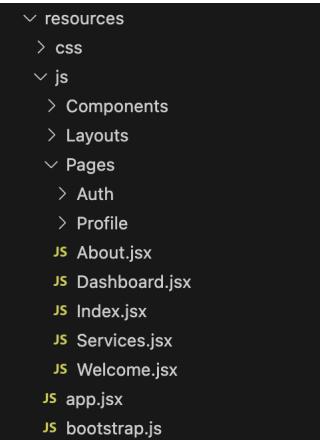
Bon, il est temps de remplir tout ça.

On commence par préparer les pages React. Le dossier `resources/js/Pages/` est déjà en place : ajoutez simplement trois fichiers vides `Index.jsx`, `Services.jsx` et `About.jsx`. Nous y mettrons du contenu plus tard — pour l'instant ils servent juste de cibles à `Inertia`.

Ensuite, il nous faut un **controller** pour relier tout ça au backend. Pour le créer, tapez dans le terminal :⁹

```
sail artisan make:controller PagesController
```

Le fichier généré se trouve dans `app/Http/Controllers`. On y ajoutera trois méthodes (`index`, `services`, `about`) qui pointeront vers nos composants React.



9. `sail artisan make:` permet de générer rapidement des fichiers (controllers, modèles, etc.). Comme nous utilisons Laravel Sail , on précède `artisan` de `sail`.

```
app > Http > Controllers > PagesController.php > ...
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Inertia\Inertia;
6
7  class PagesController extends Controller
8  {
9      public function index()
10     {
11         return Inertia::render('Index');
12     }
13
14     public function services()
15     {
16         return Inertia::render('Services');
17     }
18
19     public function about()
20     {
21         return Inertia::render('About');
22     }
23 }
24
```

FIGURE 2 – PagesController

Pour avoir un premier rendu (même minimal), déposons un simple titre dans chacun des trois composants React créés plus haut. Cela nous permettra de vérifier que les routes affichent bien la bonne page.

```
resources > js > Pages > About.jsx > ...
1  export default function About() {
2      return <h1>À propos</h1>;
3  }
4
resources > js > Pages > Services.jsx > ...
1  export default function Services() {
2      return <h1>Nos Services</h1>;
3  }
4
resources > js > Pages > Index.jsx > ...
1  export default function Index() {
2      return <h1>Accueil</h1>;
3  }
4
```

FIGURE 3 – Contenu minimal des composants React : About (gauche), Services (milieu) et Index (droite).

Enfin, pour pouvoir admirer le fruit de votre labeur, créez les routes correspondantes dans `routes/web.php`. On déclare notre controller, puis on associe chaque URL à la bonne méthode. Résultat : aller sur / affiche la page d'accueil, /services la page des services, et /about la page « À propos ». De plus, on associe un nom à chaque route de sorte à ce qu'on puisse naviguer entre les pages à la section ??.

```
1  <?php
2
3  use Illuminate\Support\Facades\Route;
4  use App\Http\Controllers\PagesController;
5
6  Route::get('/', [PagesController::class, 'Index'])->name('Index');
7  Route::get('/services', [PagesController::class, 'Services'])->name('Services');
8  Route::get('/about', [PagesController::class, 'About'])->name('About');
```

FIGURE 4 – routes/web.php

Comme toujours, le premier argument d'une route est l'adresse, et le second indique quel controller et quelle méthode exécuter. Ici, chaque méthode retournera `Inertia::render('NomDePage')`, ce qui affichera le composant React correspondant.

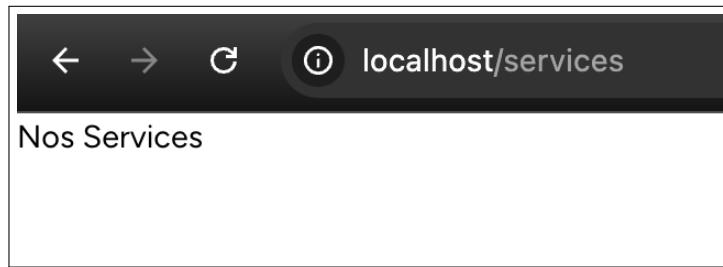


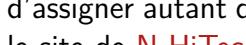
FIGURE 5 – Affichage minimal de la page Services via React + Inertia.

C'est basique, pas très joli — et c'est normal. À la prochaine section, on ajoute un layout React commun et on commence à styliser l'ensemble.

3 Tailwind & CSS

3.1 Qu'est-ce que le CSS ?

“De la même façon que HTML, CSS¹⁰ n'est pas vraiment un langage de programmation. C'est un langage de feuille de style, c'est-à-dire qu'il permet d'appliquer des styles sur différents éléments sélectionnés dans un document HTML”. (developer.mozilla.org). Cela signifie que le CSS  est le langage utilisé pour décrire comment chaque élément HTML  doit être affiché. Cela va de la taille et couleur du texte à la création de navbar, buttons, tables, etc... en passant par diverses animations simples ou plus complexes.

Le langage suit la philosophie suivante : à chaque sélecteur, on associe des propriétés. Les sélecteurs peuvent être des tags HTML  eux-mêmes, des class, id, ou d'autres choses. La bonne pratique est de styliser un maximum de composants en créant une multitude de class ayant chacune une tâche spécifique (taille, couleur, etc) afin d'obtenir une structure générale et modulaire, et ensuite d'assigner autant de class que l'on veut aux tags HTML  que l'on souhaite modifier. Par exemple, le site de N-HiTec  utilise en grande partie du rouge bordeaux. Il vient alors de créer une classe CSS appropriée qu'on peut réutiliser pour de multiples composants différents.



```
1 .nhitec-red {  
2   color: #a70025;  
3 }
```

FIGURE 6 – Exemple de classe CSS  pour la couleur N-HiTec.

10. Cascading Style Sheets



N-HiTec

Allée de la Découverte 10, 4000 Liège, Belgium
nhitec.com | info@nhitec.com

Explication du code

- .nhitec-red : sélecteur, nom de la classe CSS.
- color : propriété, en l'occurrence la couleur.
- #a70025 : valeur, ici écrite en code hexadécimal.

3.2 Pourquoi utiliser Tailwind ?

Créer un CSS pour chaque propriété d'un composant peut prendre énormément de temps. C'est pour cela que de nombreux *frameworks front-end*¹¹ existent afin d'amener de nombreuses class et plugins Javascript prédefinis. En l'occurrence, nous allons utiliser Tailwind, qui est un *framework* utilisé pour construire des sites de manière *responsive*¹² rapidement et facilement.

Pour plus de renseignementss et pour découvrir les fonctionnalités de Tailwind, rendez-vous sur [Doc Tailwind](#). Dans votre futur, vous utiliserez énormément de class Tailwind. Il est donc important de se familiariser avec rapidement, par exemple en lisant la documentation des class que vous ne connaissez pas, même si c'est fort déroutant au début afin que cela roule tout seul à moyen terme.

3.3 Installation

Pour installer Tailwind, la version 12 de Laravel nous facilite grandement la tâche car le *framework* est directement intégré lors d'une nouvelle création de projet. Il n'y a donc rien à faire.

Cependant, nous allons ajouter notre CSS contenant la couleur de notre chère et tendre JE. Pour l'instant, le dossier resources/css ne possède qu'un seul fichier app.css qui inclut la bibliothèque de Tailwind. Nous allons, dans ce même dossier, créer un fichier nhitec.css qui va accueillir notre couleur créée précédemment. Ajoutons-y le contenu de la figure ???. Pour que notre fichier nhitec.css soit accessible par nos différentes vues, nous allons l'inclure dans le fichier resources/js/types/app.jsx comme à la figure ???

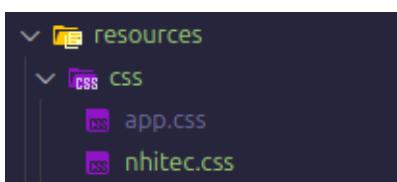
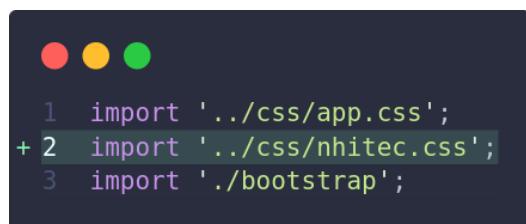


FIGURE 7 – Dossier contenant les fichiers .css



```
import './css/app.css';
+ import './css/nhitec.css';
import './bootstrap';
```

FIGURE 8 – app.jsx

11. le *front-end*, en opposition au *back-end*, désigne tout ce qui constitue l'interface visible par l'utilisateur (ex. : page web, images, etc...)

12. *responsive* signifie que le site/composant adapte son rendu en fonction de la taille de l'écran, du format etc, ce qui est quand même très important.

3.4 Utilisation

Comme expliqué plus haut, Tailwind  nous fournit une multitude de class que nous pouvons utiliser pour styliser nos tags HTML . Modifions donc nos views About.jsx et Index.jsx comme aux FIGURES ??&??.

```

1 export default function About() {
2   return (
3     <main className="max-w-3xl mx-auto p-6">
4       <div className="text-center">
5         <h1 className="text-3xl my-4 font-semibold nwhitec-red">À propos</h1>
6         <p>Cette application utilise Inertia.js et React côté front, avec Laravel côté back.</p>
7       </div>
8     </main>
9   );
10 }
11
12 <div className="mt-6 rounded-md border p-4 bg-gray-50">
13   <h2 className="text-xl font-semibold mb-2">Objectif du tuto:</h2>
14   <p>Montrer très simplement comment structurer des pages et écrire un peu de HTML/CSS sans complexifier.</p>
15   </div>
16   <div className="text-center mt-6">
17     <a href="#">Retour à l'accueil</a>
18   </div>
19 </div>
20 </main>
21 );
22 }

```

FIGURE 9 – About.jsx

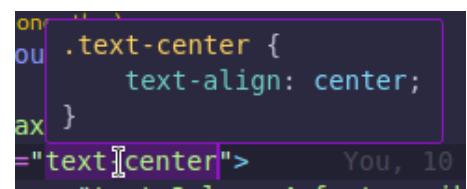
```

1 export default function Index() {
2   return (
3     <main className="max-w-3xl mx-auto p-6">
4       <div className="text-center">
5         <h1 className="text-3xl my-4 font-semibold mb-2 nwhitec-red">Accueil</h1>
6         Bienvenue sur ce petit tuto Laravel + Inertia (React).
7       </div>
8       <div className="flex items-center justify-center gap-3">
9         <a className="rounded-md border p-1 bg-gray-50" href="#about">Voir "À propos"</a>
10        <a className="rounded-md border p-1 bg-gray-50" href="#services">Voir les services</a>
11      </div>
12    </main>
13  );
14 }
15
16

```

FIGURE 10 – Index.jsx

Par exemple, dans la FIGURE ??, className="text-center" assigne la classe text-center à l'élément. Astuce : En passant la souris sur la classe Tailwind , vous verrez apparaître la vraie classe CSS  ¹³ permettant de centrer un élément dans son conteneur.



Pour la page des services, nous allons introduire une nouvelle mécanique : l'**affichage dynamique** c-à-d passer des données du back-end vers le front-end. ¹⁴ Pour le moment, nous n'allons pas encore nous embêter avec la base de données, nous allons simplement voir comment la mécanique de base fonctionne. Rappelez-vous de la Section ??, ce sont les controllers qui s'occupent de manipuler les données avant d'afficher une view. Dès lors, c'est dans la fonction services() de PagesController.php que nous allons ajouter des choses : \$titlefromcontroller et \$services sont 2 variables, et nous les passons à la view par l'intermédiaire du []. Ensuite, nous pouvons utiliser les variables title et services dans la view concernée.

```

14 public function services()
15 {
16   $titlefromcontroller = "Nos Services";
17   $services = ["Programmation web", "Introduction à la gestion d'entreprise", "De nouvelles rencontres", "Mais surtout, beaucoup de fun"];
18 }
19 return Inertia::render('Services', [
20   'title' => $titlefromcontroller,
21   'services' => $services
22 ]);

```

FIGURE 11 – PagesController.services

13. Grâce à une extension incluse dans le pack N-HiTec - Op&Ex

14. C'est comme ça que votre nom d'utilisateur est affiché sur les réseaux sociaux. Le nom d'utilisateur est récolté dans la base de données et envoyé au front-end qui l'affiche sur l'interface utilisateur.



```

1  export default function Services({ title, services }) {
2    return (
3      <main className="max-w-3xl mx-auto p-6">
4        <div className="text-center">
5          <h1 className="text-3xl my-4 font-semibold nhitec-red">{title}</h1>
6        </div>
7        <div className="rounded-md border p-4 bg-white">
8          <ul className="list-disc pl-6 space-y-1">
9            {services.map((service, index) => (
10              <li className="text-lg text-gray-700" key={index}>{service}</li>
11            )))
12          </ul>
13        </div>
14        <div className="text-center mt-6">
15          <a className="rounded-md border p-1 bg-gray-50 text-gray-500 href="">Retour à l'accueil</a>
16        </div>
17      </main>
18    );
19  }
20}
21

```

FIGURE 12 – Services.jsx

Qu'est-ce que c'est que tout ça ? Décomposons tout cela.

Dans la Section ?? nous avons pris connaissance des avantages du format .jsx. Celui-ci nous apporte les commandes {}¹⁵ et .map¹⁶.

De plus, nous découvrons ici trois nouveaux tags HTMLjsx :

1. <a> est un tag permettant la création d'un lien vers une autre URL, que l'on place dans l'attribut href. Au lieu de taper l'URL d'une route, Inertia ➔ nous permet d'optimiser l'écriture en utilisant la commande route('nomdelaroute') afin d'obtenir l'URL en question. {...} permet ensuite de l'afficher dans le href.
2. est un tag signifiant la création d'une liste.
3. représente un élément d'une liste.

Et voilà ! Maintenant, il suffit de taper npm run dev¹⁷(si ce n'était pas déjà fait) pour admirer le résultat¹⁸ (voir FIGURE ??). C'est déjà vachement mieux, non ?

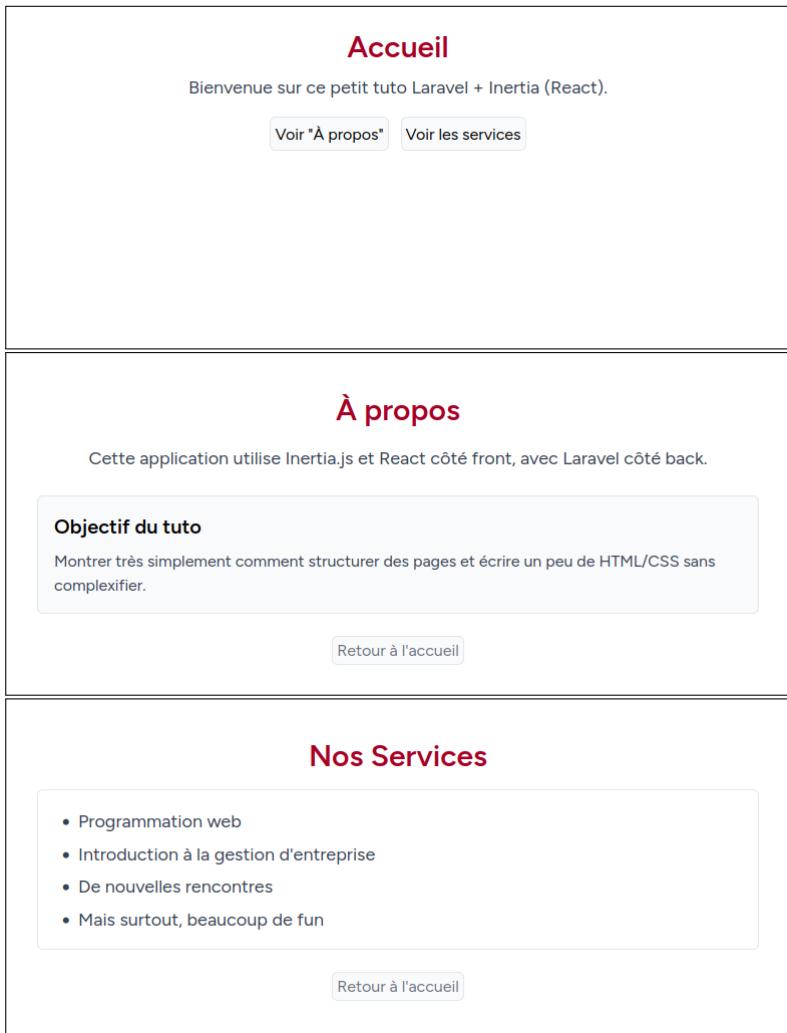
C'est bien beau, mais jusque ici le seul moyen de naviguer entre les pages est de rentrer leur URL ou revenir à l'accueil, ce qui n'est ma foi pas très pratique. Remédions à cela avant de passer à la suite.

3.5 Navbar

Comme son nom l'indique, elle sert à naviguer entre les pages. Cependant, c'est un gros morceau qui utilise beaucoup de class Tailwindcss, donc il va falloir s'accrocher.

Heureusement pour nous, React jsx inclut une bibliothèque de composants prêts à l'emploi¹⁹. Dans notre exemple, nous allons utiliser ResponsiveNavLink.jsx.

-
- 15. Cette commande agit comme un printf() en C. Elle permet d'afficher le contenu de la variable en argument.
 - 16. Boucle for classique, pour itérer sur un tableau.
 - 17. Cette commande permet de compiler le CSScss et Javascriptjs en créant un mini serveur localement. Cette commande utilisée lors du DEVELOPPEMENT DU SITE permet d'appliquer les modifications apportées à des fichiers rapidement sans devoir refresh la page. Pour compiler tout ça en production, il faut utiliser npm run build.
 - 18. L'url à entrer dans la barre de recherche dépend de la variable APP_URL se trouvant dans le .env.
 - 19. Ceux-ci se trouvent dans resources/js/Components



The figure consists of three vertically stacked screenshots of a web application. The top screenshot shows the 'Accueil' (Home) page with a red header, a white main area containing text and two buttons ('Voir "À propos"' and 'Voir les services'), and a light gray footer. The middle screenshot shows the 'À propos' (About) page with a red header, a white main area containing text and a box labeled 'Objectif du tuto' with a description, and a light gray footer. The bottom screenshot shows the 'Nos Services' (Our Services) page with a red header, a white main area containing a bulleted list of services, and a light gray footer. Each screenshot has a small button at the bottom left labeled 'Retour à l'accueil'.

(a) <http://example/>

(b) <http://example/about>

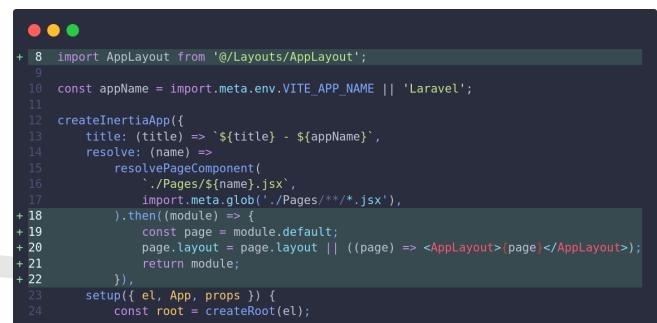
(c) <http://example/services>

FIGURE 13 – 3 pages créées jusqu'à présent et stylisées avec Tailwind .

D'abord, dans le dossier `resources/js/Layouts`, créez un fichier `AppLayout.jsx`. Remplissez ce fichier avec le contenu de la FIGURE ??.

Ensuite, il faut ajouter notre `navbar` à notre `layout` afin qu'elle apparaisse sur toutes nos pages. Pour ce faire, rien de plus simple. Ajoutez ce bout de code dans `resources/js/app.tsx`.

Pour le reste, je vous invite à lire ce que font chaque class et de jeter un œil sur [la doc Tailwind](#) sur les `navbars`. Bien que ça soit indigeste lors d'une première lecture, ça l'est beaucoup moins que si nous devions analyser les class une par une



```
+ 8 import AppLayout from '@/Layouts/AppLayout';
9
10 const appName = import.meta.env.VITE_APP_NAME || 'Laravel';
11
12 createInertiaApp({
13   title: (title) => `${title} - ${appName}`,
14   resolve: (name) =>
15     resolvePageComponent(
16       `./Pages/${name}.jsx` ,
17       import.meta.glob('./Pages/**/*.jsx'),
18     ).then((module) => {
19       const page = module.default;
20       page.layout = page.layout || ((page) => <AppLayout>(page)</AppLayout>);
21
22     }),
23     setup({ el, App, props }) {
24       const root = createRoot(el);
```

Néanmoins, il reste un point important à aborder : le *responsive design* ou *design adaptatif*, c'est le fait de modifier la page web en fonction de la taille de la fenêtre pour convenir à tous les supports.



Ceci est un document interne. Ne pas diffuser.



```

1 import ResponsiveNavLink from '@/Components/ResponsiveNavLink';
2 import { Link, usePage } from '@inertiajs/react';
3
4 export default function AppLayout({ children }) {
5   const { component } = usePage();
6   const isActive = (name) => component === name;
7
8   return (
9     <div className="min-h-screen bg-gray-100">
10       <header className="bg-white border-b">
11         <div className="mx-auto max-w-5xl px-4 py-3 flex items-center justify-between">
12           <Link href="/" className="font-semibold text-lg nhitec-red">
13             Tuto Laravel
14           </Link>
15         </div>
16         <nav className="bg-gray-50">
17           <div className="mx-auto max-w-5xl px-2 py-1 space-y-1 sm:flex sm:space-y-0 sm:space-x-2">
18             <ResponsiveNavLink className="nhitec-red" href={route('Index')} active={isActive('Index')}>
19               Accueil
20             </ResponsiveNavLink>
21             <ResponsiveNavLink className="nhitec-red" href={route('About')} active={isActive('About')}>
22               À propos
23             </ResponsiveNavLink>
24             <ResponsiveNavLink className="nhitec-red" href={route('Services')} active={isActive('Services')}>
25               Services
26             </ResponsiveNavLink>
27           </div>
28         </nav>
29       </header>
30       <main className="mx-auto max-w-5xl px-4 py-6">{children}</main>
31     </div>
32   );
33 }

```

FIGURE 14 – AppLayout.jsx

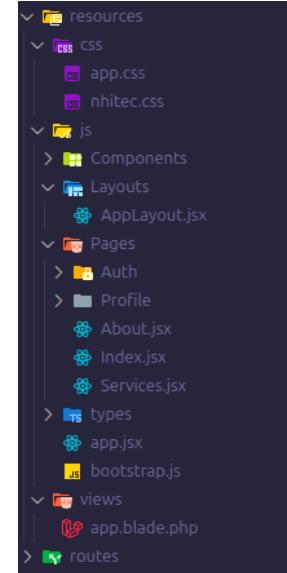


FIGURE 15 –
Structure du projet



FIGURE 16 – Navbar sur un écran de largeur > 640px



FIGURE 17 – Navbar vue depuis un téléphone (largeur < 640px)²⁰

20. La couleur rouge est obtenue en remplaçant les class CSS  indigo par red dans ResponsiveNavLink.tsx

4 Base de données

Nous allons créer une base de données en utilisant l'exemple de posts sur un blog.

4.1 PhpMyAdmin

PhpMyAdmin  est une interface permettant à des ignares comme nou... comme vous* de manipuler et de visualiser le contenu des bases de données facilement sans connaissances en MySQL. Normalement, la configuration a été effectuée lors de la formation précédente et vous devriez pouvoir y accéder en vous rendant sur <http://example:8080>.

4.2 Models

Comme dit dans la Section ??, le model ²¹ est l'objet qui nous permettra d'interagir avec les *posts* stockés dans notre base de données ²². Pour le créer, tapez `sail artisan make:model Post -m`, et observez la création d'un fichier `Post.php` dans `app/Models/`. Pour le moment, la classe (au sens PHP  du terme) est vide, mais nous pouvons ajouter des méthodes ²³ spécifiques à ce model, ce qui, nous le verrons, est très pratique.

4.3 Migrations : comme les oiseaux ?

Une migration est un fichier qui permet de définir les tables de notre base de données. Une table est grossièrement un type de donnée que la base de données va stocker. Par exemple, la liste de tous les utilisateurs est une table, tout comme les *posts* que nous allons créer. Chaque table contient un certain nombre de columns qui elles sont les données stockées en tant que telles. En l'occurrence, notre table de *posts* doit contenir une column pour le titre d'un *post* et une pour son contenu en lui-même ²⁴.

Pour créer notre migration, tapez rien du tout car la migration a été créée en même temps que notre model grâce au `-m` ! Elle se trouve dans `database/migration/xxxx_xx_xx_xxxxxxx_create_posts_table.php`.
Ensuite, remplissez-la comme à la FIGURE ?? ²⁵ :

Analysons tout à :

- function `up()` et function `down()` : La première est exécutée quand on souhaite créer les tables à l'intérieur tandis que la deuxième est exécutée lorsqu'on souhaite supprimer les tables de la base de données. Pour l'utilisation simple que nous faisons des migrations, pensez à `down` tout ce que vous `up`-per.

21. Rien à voir avec Kendall Jenner, reste concentré sur la formation.

22. pour chaque nouvel objet, on aura un model correspondant

23. C'est juste un nom pour désigner une fonction se trouvant dans une classe en programmation orientée objet (POO). cf. [INFO0062 - Object-Oriented Programming](#) de Boiglot.

24. Tous ces termes sont compliqués à décrire avec des mots, mais sont en réalité très intuitifs quand on imagine les données affichées dans un grand tableau à double entrée.

25. lignes 16 et 17, de rien

- Schema::create('posts', ... en gros, c'est la fonction utilisée pour créer la table 'posts', et les \$table-> qui suivent permettent de définir chaque column de la table créée.
- id() : l'id est ce qu'on appelle la Primary Key. Unique pour chaque row²⁶, il permet d'identifier chaque élément de donnée. Il se trouve par défaut sur chaque table nouvellement créée.
- string() permet de créer une column de type string, de taille 255. La taille est modifiable en ajoutant un nombre <255 en second argument.
- mediumtext() permet de créer une column de 16.777.215 caractères (oula).
- timestamps() ajoute une date de création et de modification à la table. Ces deux columns sont également ajoutées par défaut.

Quelques remarques supplémentaires :

1. Par défaut, les columns doivent obligatoirement posséder une valeur.
2. On peut ajouter de nombreux paramètres aux columns pour modifier leur comportements (exemple : ->nullable() pour leur permettre d'être vide).
3. Les types des column correspondent chacune à un type de valeur de MySQL, le "vrai" langage pour communiquer avec les base de données duquel Laravel  nous protège grâce aux models, migrations, tables que nous venons de voir.

Encore une fois, parcourir la doc officielle de Laravel  permet d'en apprendre beaucoup plus que ce que le tutoriel ne pourra jamais vous apprendre !

4.4 Migrations : exécution

Bon, après tant de blabla, passons à l'action. Mais avant cela, Laravel  nous embête (pour une fois). Afin de n'avoir aucune erreur en exécutant la migration, il va falloir ajouter ces lignes dans app/Providers/AppServiceProvider.php :

```
4
5  use Illuminate\Support\ServiceProvider;
6  use Illuminate\Support\Facades\Schema;
7
8  class AppServiceProvider extends ServiceProvider
```

```
database > migrations > 2023_07_22_194605_create_posts_table.php > ...
1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  return new class extends Migration
8  {
9      /**
10     * Run the migrations.
11     */
12    public function up(): void
13    {
14        Schema::create('posts', function (Blueprint $table) {
15            $table->id();
16            $table->string('title');
17            $table->mediumText('body');
18            $table->timestamps();
19        });
20    }
21
22    /**
23     * Reverse the migrations.
24     */
25    public function down(): void
26    {
27        Schema::dropIfExists('posts');
28    }
29};
30
```

FIGURE 18 – Exemple très simple de migration

```
21  public function boot(): void
22  {
23      Schema::defaultStringLength(191);
24  }
25 }
```

Voilà ! maintenant tapez sail artisan migrate:fresh (où fresh signifie que tout ce qui existait avant est supprimé et --seed permet d'exécuter les seeders vus dans la Section (WIP)) et, si tout va bien, vous verrez maintenant cela en allant dans PhpMyAdmin  :

26. une row est un élément de donnée dans une table



4.5 PostsController

Pour gérer ces posts, nous allons bien entendu avoir besoin d'un controller. Comme ce genre de données va avoir des manipulations basiques très communes (création, liste, affichage, modification, suppression,...), il existe un certain type de controller permettant de nous faire gagner du temps : le resource controller. Tapez donc

```
sail artisan make:controller PostsController --resource
```

pour en créer un.

Ensuite, il faut évidemment définir des nouvelles routes. Une seule ligne toute simple nous permet de générer en réalité 7 routes différentes que nous verrons petit à petit. Ajoutez donc ces 2 lignes dans `routes/web.php` :

```
21 Route::get('/about', [PagesController::class, 'about'])->name('about');
22
23 | Route::resource('posts', PostsController::class);
```

FIGURE 19 – `routes/web.php`

```
5  use App\Http\Controllers\PagesController;
6  use App\Http\Controllers\PostsController;
7
```

FIGURE 20

Notez que en donnant '`posts`' en argument à `resource()`, celui-ci fait automatiquement lien avec le model Post²⁷.

Petite parenthèse avant de continuer : un exemple de route générée par la commande de la FIGURE ?? est :

`Route::get('/posts/{post}', [PostsController::class, 'show'])->name('posts.show')`

le `{...}` dans l'URL est une sorte de paramètre dans l'URL, qui permet de passer des informations au controller. En effet, chaque paramètre dans l'URL sera passé (si on le souhaite) en argument à la fonction du controller appellée par cette route, dans le même ordre d'apparition que dans l'URL. Ce paramètre est extrêmement utile comme nous le verrons à la Section ???. Pour plus d'informations sur les routes générées par cette commande, voyez [ceci](#). Fin de la parenthèse.

Bon, maintenant, il faut remplir ce controller et créer les views qui vont avec. Commençons par les plus simples, `index()` et `show()`.

4.5.1 index

Cette méthode est utilisée pour afficher la liste de tous les Posts créés.

27. Pour comprendre comment Laravel  fait pour être si intelligent, jetez un oeil à [ceci](#).

Avec cet exemple simple, on comprend vite à quel point il est simple d'interagir avec la base de données par l'intermédiaire des `models`. Nos posts sont stockés sous forme d'un array d'objets PHP ^{PHP}, contenant les champs que nous avons spécifiés dans la `migration`.

Ensuite, nous allons créer un nouveau dossier `Posts` dans

`resources/js/Pages` et ajouter un fichier dedans appelé

`Index.jsx`. Il ne reste "plus qu'à" le remplir avec le contenu de la FIGURE ??.

Ici, tout est déjà connu mis à part le `posts.length === 0` ? (X) : (Y), qui vérifie s'il y a un post : si non, il affiche X sinon Y²⁸. Ensuite, il y a le "." permettant d'accéder aux différents champs des objets que sont nos `$post`.

Enfin, il ne nous reste plus qu'à ajouter un bouton à notre navbar²⁹ afin d'accéder à cette page.

```

1  export default function PostsIndex({ posts = [] }) {
2    return (
3      <div className="max-w-3xl mx-auto p-6">
4        <div className="text-center my-4">
5          <h1 className="text-3xl font-semibold nhtec-red">Publications</h1>
6        </div>
7
8        <div className="rounded-md border bg-white">
9          {posts.length === 0 ? (
10            <div className="p-4 text-gray-500">Aucune publication n'a été trouvée.</div>
11          ) : (
12            <ul className="divide-y">
13              {posts.map((p) => (
14                <li key={p.id} className="p-4 flex items-center justify-between">
15                  <a href={route('posts.show', p.id)}>
16                    {p.title}
17                  </a>
18                  <div className="text-xs text-gray-500">#{p.id}</div>
19                  <a href={route('posts.show', p.id)}>
20                    Voir
21                  </a>
22                </li>
23              )));
24            )
25          )
26        </ul>
27      </div>
28
29      <div className="text-center mt-6">
30        <a href={route('posts.create')}>
31          Créer une publication
32        </a>
33      </div>
34    );
35  }
36}
37

```

FIGURE 21 – `Index.jsx`

```

22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37

```

4.5.2 show

Même chose que pour la page précédente, il faut remplir la fonction `show()`.

Ici, nous utilisons ce que j'ai expliqué en dessous de la FIGURE ?? . La fonction `show()` prend en argument `$post`. Bah oui, ce qui est logique car pour afficher un post en particulier, il faut donner à Laravel  le `$post` en question.

```

36  public function show(Post $post)
37  {
38    return Inertia::render('Posts/Show', [
39      'post' => $post->only(['id', 'title', 'body', 'created_at']),
40    ]);
41  }

```

28. Il s'agit d'un opérateur ternaire, comme en C, permettant de faire un `if...else` en une seule ligne.

29. Dans `Layouts/AppLayout.jsx` (juste de rien pour ça)

Enfin, la view correspondante peut être créée au nom et emplacement resources/js/Pages/Posts>Show.jsx

```

1  export function formatDateFrBE(value) {
2    if (!value) return '';
3    try {
4      return new Date(value).toLocaleString('fr-BE', {
5        day: '2-digit',
6        month: '2-digit',
7        year: 'numeric',
8        hour: '2-digit',
9        minute: '2-digit',
10       });
11    } catch {
12      return value;
13    }
14  }
15
16 export default function PostsShow({ post }) {
17  if (!post) return null;
18  return (
19    <div className="max-w-3xl mx-auto p-6">
20      <div className="text-center mb-4">
21        <h1 className="text-3xl font-semibold nhitec-red">{post.title}</h1>
22        <div className="text-xs text-gray-500">#{post.id}</div>
23      </div>
24
25      <article className="rounded-md border bg-white p-4">
26        <p className="text-gray-800 whitespace-pre-line">{post.body}</p>
27        <div className="text-xs text-gray-500 mt-2">Publié le {formatDateFrBE(post.created_at)}</div>
28      </article>
29
30      <div className="text-center mt-6">
31        <a className="rounded-md border p-1 bg-gray-50 text-gray-700" href={route('posts.index')}>
32          Retour à la liste
33        </a>
34      </div>
35    </div>
36  );
37}
38

```

Ici, on a créé une fonction `formatDateFrBE()` qui permet d'afficher la date au format belge au lieu du format US.

Hormis ceci, R.A.S. en termes de nouveautés donc nous pouvons maintenant passer à la création de posts !

4.5.3 create

Cette partie est plus intéressante car elle va amener un nouveau concept, les `forms`. Jusque ici, nous n'avons jamais eu à rentrer nous-mêmes des données et à les envoyer à notre site pour qu'il fasse des choses avec, c'est exactement à cela que servent les `forms`.

Tout d'abord, remplissons la méthode `create()` du controller et créons le fichier `resources/posts/create.blade.php` comme nous en avons maintenant l'habitude.

Maintenant, il faut remplir la view :

Ici, il y a pas mal de nouveautés. On voit, en effet, apparaître trois nouveaux tags HTML  :

```

19  public function create()
20  {
21    return Inertia::render('Posts/Create');
22  }

```

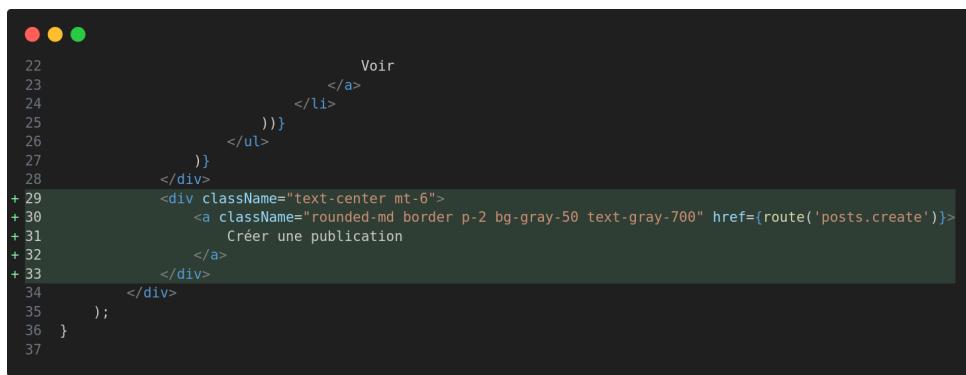
```
1 import { useForm } from '@inertiajs/react';
2
3 export default function PostsCreate() {
4     const { data, setData, post, processing, errors } = useForm({
5         title: '',
6         body: '',
7     });
8
9     const submit = (e) => {
10         e.preventDefault();
11         post(route('posts.store'));
12     };
13
14     return (
15         <div className="max-w-3xl mx-auto p-6">
16             <div className="text-center mb-4">
17                 <h1 className="text-3xl font-semibold nhitec-red">Nouvelle publication</h1>
18             </div>
19
20             <form onSubmit={submit} className="rounded-md border bg-white p-4 space-y-3">
21                 <div>
22                     <label className="block text-sm font-medium mb-1">Titre</label>
23                     <input
24                         type="text"
25                         className="w-full rounded-md border p-2"
26                         value={data.title}
27                         onChange={(e) => setData('title', e.target.value)}
28                     />
29                     {errors.title && (
30                         <div className="text-sm text-red-600 mt-1">{errors.title}</div>
31                     )}
32                 </div>
33
34                 <div>
35                     <label className="block text-sm font-medium mb-1">Contenu</label>
36                     <textarea
37                         className="w-full rounded-md border p-2 min-h-[10rem]"
38                         value={data.body}
39                         onChange={(e) => setData('body', e.target.value)}
40                     />
41                     {errors.body && (
42                         <div className="text-sm text-red-600 mt-1">{errors.body}</div>
43                     )}
44                 </div>
45
46                 <div className="flex items-center gap-2">
47                     <button className="rounded-md border px-3 py-1 bg-gray-50">
48                         Publier
49                     </button>
50                     <a className="rounded-md border px-3 py-1" href={route('posts.index')}>Annuler</a>
51                 </div>
52             </form>
53         </div>
54     );
55 }
56 }
```

1. **<form>** : Ce bloc regroupe les champs du formulaire. L'envoi est contrôlé par la fonction passée à `onSubmit`. Dans l'exemple, `submit` empêche le rechargement de la page (`e.preventDefault()`) puis envoie les données via `post(route('posts.store'))`. La fonction `post` provient du hook `useForm` et `route('posts.store')` génère l'URL de la route backend.
2. **<label>** : Sert à afficher un intitulé lisible pour l'utilisateur. Placé juste avant le champ auquel il se rapporte. Pour renforcer l'accessibilité, on pourrait ajouter un attribut `htmlFor` (et un

id correspondant sur le champ).

3. `<input type="text">` : Champ de saisie contrôlé pour le titre. La valeur affichée provient de l'état (`value={data.title}`). Chaque frappe déclenche `onChange` qui met à jour l'état via `setData('title', e.target.value)`³⁰. L'affichage conditionnel d'un message d'erreur se fait avec `errors.title`.
4. `<textarea>` : Champ de texte multi-lignes pour le contenu. Il est également contrôlé : `value={data.body}` et mise à jour via `setData('body', e.target.value)`. Le message d'erreur associé utilise `errors.body`.

Bon, ce fût beaucoup (trop) de bla-bla, mais on va (enfin) pouvoir passer à la suite ! Promis, les trois dernières méthodes seront bien plus simples. Mais avant cela, n'oubliez pas d'ajouter un bouton sur la page `Posts/Index.jsx` pour accéder à cette page :



A screenshot of a code editor showing a snippet of JSX code. The code includes several numbered lines (22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37). Lines 29 through 33 are highlighted in green, indicating they are new additions. These lines define a `<div>` element with a specific class and a `` link labeled "Créer une publication". The rest of the code is standard JSX for rendering a list of items with "Voir" and "Modifier" buttons.

FIGURE 22 – Ajout du bouton "Créer une publication" dans `Index.jsx`

30. Au cas où, le e c'est juste pour pas écrire event au complet, voilà voilà.

4.5.4 store

On a vu à la section précédente que les données du form étaient envoyées à la route 'posts.store', or cette route redirige vers cette fonction ! C'est donc ici que nous allons stocker le post nouvellement créé.

Tout d'abord, nous allons vérifier si les 2 champs titre et message ont bien été remplis (s'ils doivent être remplis, ils sont requis ⇒ required). Pour cela, on utilise la fonction validate() de Laravel ³¹. Si la validation rate, l'utilisateur est renvoyé à la page précédente et si elle réussit, alors on continue.

L'étape suivante est de créer un nouveau post, via son model. Ensuite, on remplit ses champs avec les valeurs obtenues dans le form et on le sauvegarde dans la base de donnée. Enfin, il ne reste plus qu'à envoyer l'utilisateur quelque part (la liste des posts) et le tour est joué ! ³² Plus qu'à rajouter les fonctionnalités de modification et de suppression et puis nous en aurons terminé.

```
● ● ●
24 public function store(Request $request)
25 {
26     $data = $request->validate([
27         'title' => ['required', 'string', 'max:255'],
28         'body' => ['required', 'string'],
29     ]);
30
31     $post = Post::create($data);
32
33     return redirect()->route('posts.show', $post);
34 }
```

FIGURE 23 – Méthode store()

4.5.5 edit

Pour l'édition, c'est plutôt simple. Tout d'abord, remplissons la méthode du controller :

```
● ● ●
43 public function edit(Post $post)
44 {
45     return Inertia::render('Posts/Edit', [
46         'post' => $post->only(['id', 'title', 'body']),
47     ]);
48 }
```

FIGURE 24 – Méthode edit

31. Plus d'informations ainsi que la liste des règles de validation [ici](#).
32. La variable 'success' sera utilisée dans la Section ??

Ensuite, créons la view dédiée :

```

● ● ●
1 import { useForm } from '@inertiajs/react';
2
3 export default function Edit({ post }) {
4   const { data, setData, patch, processing, errors } = useForm({
5     title: post?.title || '',
6     body: post?.body || '',
7   });
8
9   if (!post) {
10     return <div className="max-w-3xl mx-auto p-6">Publication introuvable.</div>;
11   }
12
13   const submit = (event) => {
14     event.preventDefault();
15     patch(route('posts.update', post.id));
16   };
17
18   return (
19     <div className="max-w-3xl mx-auto p-6">
20       <div className="text-center mb-4">
21         <h1 className="text-3xl font-semibold nhitec-red">Modifier la publication {post.id}</h1>
22       </div>
23
24       <form onSubmit={submit} className="rounded-md border bg-white p-4 space-y-3">
25         <div>
26           <label className="block text-sm font-medium mb-1">Titre</label>
27           <input
28             type="text"
29             className="w-full rounded-md border p-2"
30             value={data.title}
31             onChange={(e) => setData('title', e.currentTarget.value)}
32           />
33           {errors.title && <div className="text-sm text-red-600 mt-1">{errors.title}</div>}
34         </div>
35
36         <div>
37           <label className="block text-sm font-medium mb-1">Contenu</label>
38           <textarea
39             className="w-full rounded-md border p-2 min-h-[10rem]"
40             value={data.body}
41             onChange={(e) => setData('body', e.currentTarget.value)}
42           />
43           {errors.body && <div className="text-sm text-red-600 mt-1">{errors.body}</div>}
44         </div>
45
46         <div className="flex items-center gap-2">
47           <button className="rounded-md border px-3 py-1 bg-gray-50">
48             Enregistrer
49           </button>
50           <a className="rounded-md border px-3 py-1" href={route('posts.index')}>Annuler</a>
51         </div>
52       </form>
53     </div>
54   );
55 }
56

```

FIGURE 25 – Posts/Edit.jsx

Remarquez qu'il n'y a presque aucune différence avec la view de création de post, et c'est bien normal : on peut l'utiliser quasi à l'identique à condition de mettre les valeurs du post dans les bons champs lorsque l'on accède à la page. Une différence est néanmoins à noter, la présence du `patch()`, ligne 15. Cette méthode permet de remplacer une donnée par une autre (notre post à modifier).

4.5.6 update

Lorsque le `form` est envoyé, il faut bien sûr enregistrer le post dans notre base de données. La route '`posts.update`' nous envoie donc vers la méthode `update()` que vous pouvez remplir comme ci-dessous. Notez que ici aussi, la démarche est la même que pour la création de post (et c'est logique), mis à part qu'on cherche le post à modifier au lieu d'en créer un nouveau.

```
● ● ●
50 public function update(Request $request, Post $post)
51 {
52     $data = $request->validate([
53         'title' => ['required', 'string', 'max:255'],
54         'body' => ['required', 'string'],
55     ]);
56
57     $post->update($data);
58
59     return redirect()->route('posts.show', $post);
60 }
```

FIGURE 26 – Méthode update

4.5.7 destroy

Enfin, la suppression de post. Tout d'abord, ajoutons deux boutons dans `Posts/Index.jsx`, un qui permet d'accéder à notre page de modification d'un post, et un autre qui permettra de le supprimer (FIGURE ??).

Ensuite, remplissons la méthode du controller. La mécanique est ici triviale, il suffit de sélectionner le post que l'on souhaite et ensuite de le supprimer.

Et voilà ! Nous avons désormais un système (simple) de création et gestion de posts ! Évidemment, nous pourrions rajouter énormément de fonctionnalités (commentaires, auteurs, images, ...). Nous explorerons certaines de ces options dans de futures sections, pour cette introduction, c'est déjà plus que suffisant.

```
● ● ●
62 public function destroy(Post $post)
63 {
64     $post->delete();
65
66     return redirect()->route('posts.index');
67 }
```

FIGURE 27 – Méthode destroy

A titre d'indication, voici le résultat auquel vous devriez arriver si tout fonctionne correctement :



Ceci est un document interne. Ne pas diffuser.

```

● ● ●
+ 1 import { router } from '@inertiajs/react';
2
3 export default function PostsIndex({ posts = [] }) {
4     return (
5         <div className="max-w-3xl mx-auto p-6">
6             <div className="text-center my-4">
7                 <h1 className="text-3xl font-semibold nhitec-red">Publications</h1>
8             </div>
9
10            <div className="rounded-md border bg-white">
11                {posts.length === 0 ? (
12                    <div className="p-4 text-gray-500">Aucune publication n'a été trouvée.</div>
13                ) : (
14                    <ul className="divide-y">
15                        {posts.map((p) => (
16                            <li key={p.id} className="p-4 flex items-center justify-between">
17                                <div>
18                                    <a className="nhitec-red hover:underline" href={route('posts.show', p.id)}>
19                                        {p.title}
20                                    </a>
21                                    <div className="text-xs text-gray-500">#{p.id}</div>
22                                </div>
23
24                                <div className="flex items-center gap-2">
25                                    <a className="rounded-md border px-2 py-1 text-sm" href={route('posts.show', p.id)}>
26                                        Voir
27                                    </a>
28
29                                    <a className="rounded-md border px-2 py-1 text-sm" href={route('posts.edit', p.id)}>
30                                        Modifier
31                                    </a>
32                                    <button
33                                        type="button"
34                                        className="rounded-md border px-2 py-1 text-sm"
35                                        onClick={() => {
36                                            if (confirm('Supprimer cette publication ?')) {
37                                                router.delete(route('posts.destroy', p.id));
38                                            }
39                                        }}
40                                    >
41                                        Supprimer
42                                    </button>
43                                </div>
44                            </li>
45                        ))}
46                    </ul>
47                )}
48            </div>
49
50            <div className="text-center mt-6">
51                <a className="rounded-md border p-2 bg-gray-50 text-gray-700" href={route('posts.create')}>
52                    Créer une publication
53                </a>
54            </div>
55        </div>
56    );
57 }

```

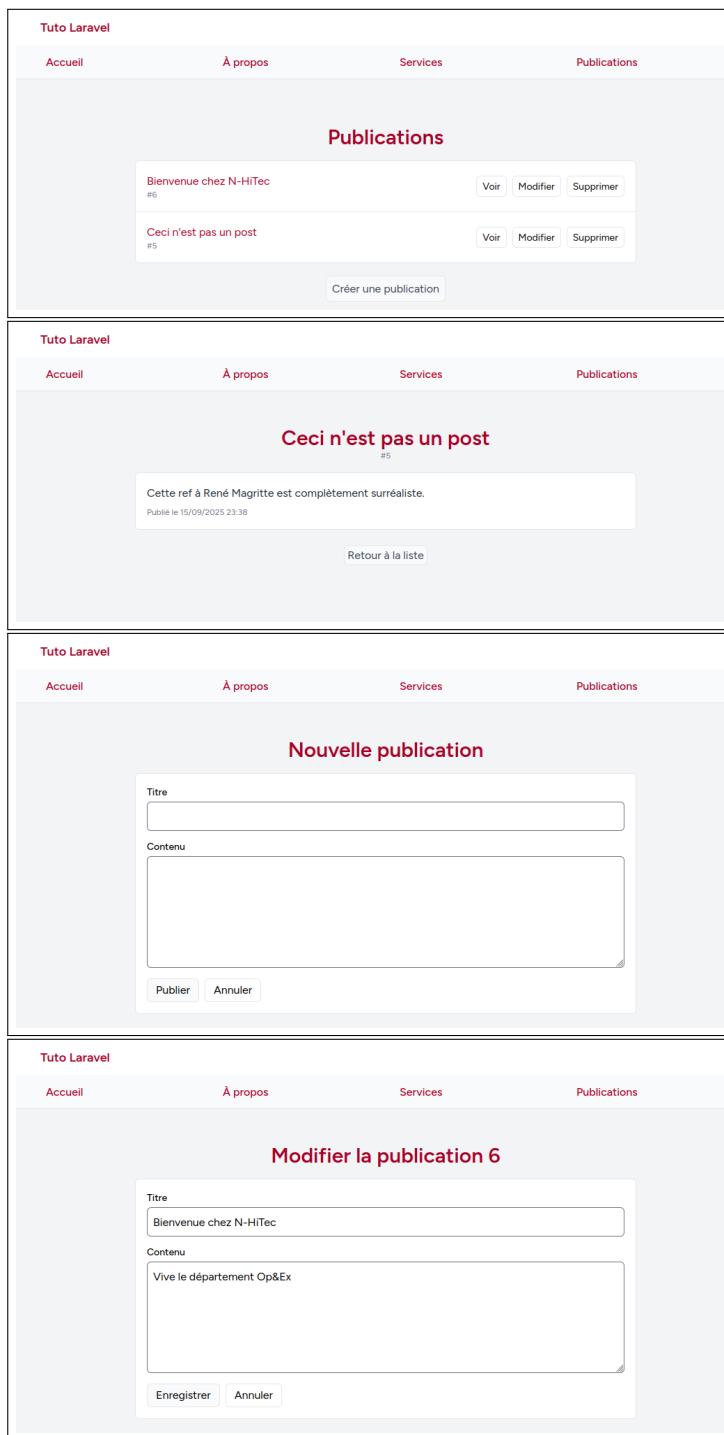
FIGURE 28 – Ajout des boutons "Modifier" et "Supprimer" dans Index.jsx



N-HiTec

Allée de la Découverte 10, 4000 Liège, Belgium
nhitec.com | info@nhitec.com

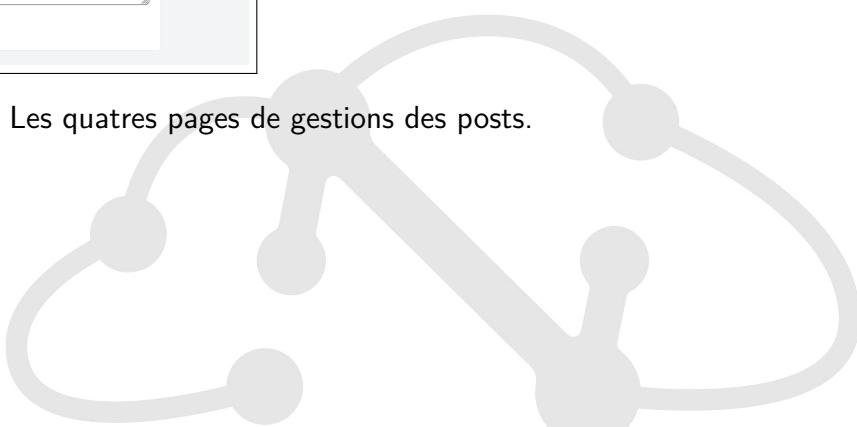
Ceci est un document interne. Ne pas diffuser.



The figure consists of four vertically stacked screenshots of a web application titled "Tuto Laravel".

- (a) <http://localhost/posts>**: Shows a list of publications. The first entry is "Bienvenue chez N-HiTec" and the second is "Ceci n'est pas un post". Each entry has "Voir", "Modifier", and "Supprimer" buttons.
- (b) <http://localhost/posts/1>**: Shows the detail view for the post "Ceci n'est pas un post". It contains the text "Cette ref à René Magritte est complètement surréaliste." and the timestamp "Publié le 15/09/2025 23:38". A "Retour à la liste" button is at the bottom.
- (c) <http://localhost/posts/create>**: Shows the form for creating a new publication. It has fields for "Titre" and "Contenu", and buttons for "Publier" and "Annuler".
- (d) <http://localhost/posts/1/edit>**: Shows the form for editing the post "Ceci n'est pas un post". The "Titre" field contains "Bienvenue chez N-HiTec" and the "Contenu" field contains "Vive le département Op&Ex". It has buttons for "Enregistrer" and "Annuler".

FIGURE 29 – Les quatres pages de gestions des posts.



5 Toasts : mi-am mi-am avec du Nutella

Non non non, rangez vos grille-pains, il ne s'agit, en l'occurrence, pas de nourriture mais bien de messages. Bah quoi c'est tout aussi important ! Ils permettent de transmettre de l'information à nos utilisateurs après une action de leur part³³. En effet, vous avez peut-être remarqué un gros inconvénient de notre système : rien ne nous dit qu'un post est créé/modifié/supprimé après une action. C'est pourquoi, nous allons ajouter nos fameux toasts³⁴

Créez donc un fichier `resources/js/Layouts/Toast.jsx` et remplissez-le comme sur la figure de droite.

Ce composant récupère les messages flash (`success` / `error`) envoyés par Laravel via Inertia et les affiche. Il montre le message, le cache automatiquement après 4 secondes ou immédiatement si on clique sur "Fermer". S'il n'y a rien à afficher, il ne rend rien.

Maintenant, il faut faire parvenir la variable `success` et `error` via notre

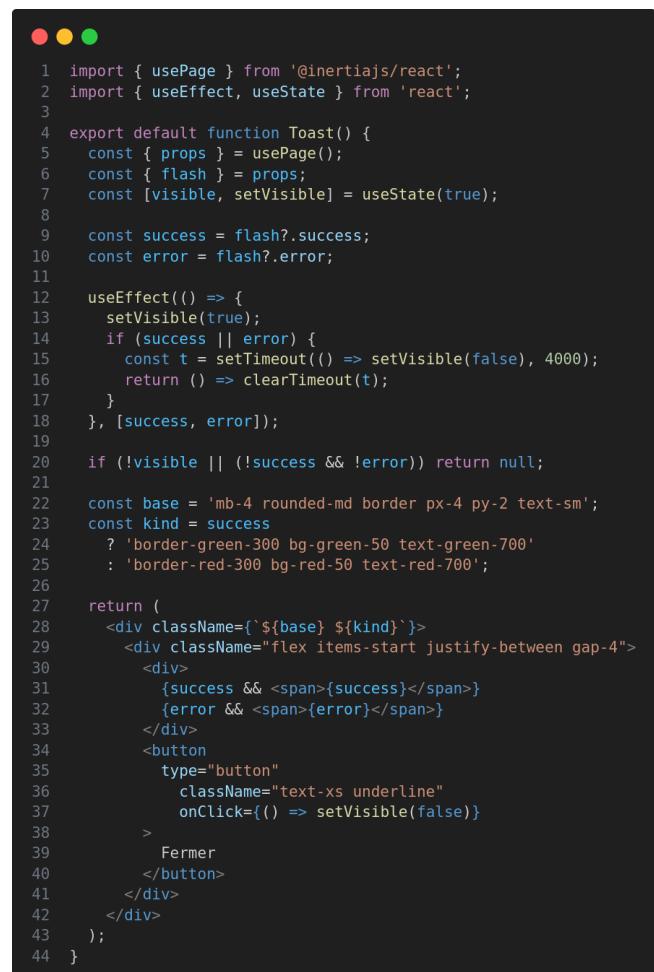
`PostsController`³⁵. Ajoutez-y donc ceci :



```

24     public function store(Request $request)
25     {
26         $data = $request->validate([
27             'title' => ['required', 'string', 'max:255'],
28             'body' => ['required', 'string'],
29         ]);
30
31         $post = Post::create($data);
32
33         return redirect()
34             ->route('posts.show', $post->id)
35             ->with(['success' => 'Publication créée avec succès.']);
36     }
37
38     public function show(Post $post)
39     {
40         return Inertia::render('Posts/Show', [
41             'post' => $post->only(['id', 'title', 'body', 'created_at']),
42         ]);
43     }
44
45     public function edit(Post $post)
46     {
47         return Inertia::render('Posts/Edit', [
48             'post' => $post->only(['id', 'title', 'body']),
49         ]);
50     }
51
52     public function update(Request $request, Post $post)
53     {
54         $data = $request->validate([
55             'title' => ['required', 'string', 'max:255'],
56             'body' => ['required', 'string'],
57         ]);
58
59         $post->update($data);
60
61         return redirect()
62             ->route('posts.show', $post->id)
63             ->with(['success' => 'Publication mise à jour.']);
64     }
65
66     public function destroy(Post $post)
67     {
68         $post->delete();
69
70         return redirect()
71             ->route('posts.index')
72             ->with(['success' => 'Publication supprimée.']);
73     }
74 }
```

FIGURE 31 – PostsController.php



```

1 import { usePage } from '@inertiajs/react';
2 import { useEffect, useState } from 'react';
3
4 export default function Toast() {
5     const { props } = usePage();
6     const { flash } = props;
7     const [visible, setVisible] = useState(true);
8
9     const success = flash?.success;
10    const error = flash?.error;
11
12    useEffect(() => {
13        setVisible(true);
14        if (success || error) {
15            const t = setTimeout(() => setVisible(false), 4000);
16            return () => clearTimeout(t);
17        }
18    }, [success, error]);
19
20    if (!visible || (!success && !error)) return null;
21
22    const base = 'mb-4 rounded-md border px-4 py-2 text-sm';
23    const kind = success
24        ? 'border-green-300 bg-green-50 text-green-700'
25        : 'border-red-300 bg-red-50 text-red-700';
26
27    return (
28        <div className={`${base} ${kind}`}>
29            <div className="flex items-start justify-between gap-4">
30                <div>
31                    {success && <span>{success}</span>}
32                    {error && <span>{error}</span>}
33                </div>
34                <button
35                    type="button"
36                    className="text-xs underline"
37                    onClick={() => setVisible(false)}
38                >
39                    Fermer
40                </button>
41            </div>
42        </div>
43    );
44 }
```

FIGURE 30 – Layouts/Toast.jsx

33. Une récompense qu'on attend pas mais qui fait plaisir, comme recevoir un ... toast au Nutella

34. Petits messages temporaires de confirmation, erreur, information et plus encore.

35. app/Http/Controller/PostsController.php, de rien pour ça.

Ceci est un document interne. Ne pas diffuser.

Avant de tester tout ça, il nous reste à permettre Inertia  de partager ces variables. Ajoutez ceci dans le middleware³⁶ app/Http/Middleware/HandleInertiaRequest.php.

Ainsi, il nous reste à ajouter les toasts dans notre layout pour les afficher à toutes les pages :

```

1 import ResponsiveNavLink from '@Components/ResponsiveNavLink';
2 import { Link, usePage } from '@inertiajs/react';
+ 3 import Toast from './Toast';
4
5 export default function AppLayout({ children }) {
6   const { component } = usePage();
7   const isActive = (name) => component === name;
8
9   return (
10     <div className="min-h-screen bg-gray-100">
11       <header className="bg-white border-b">
12         <div className="mx-auto max-w-5xl px-4 py-3 flex items-center justify-between">
13           <Link href={route('Index')} className="font-semibold text-lg nwhitec-red">
14             Tuto Laravel
15           </Link>
16         </div>
17         <nav className="bg-gray-50">
18           <div className="mx-auto max-w-5xl px-2 py-1 sm:flex sm:space-y-0 sm:space-x-2">
19             <ResponsiveNavLink className="nwhitec-red" href={route('Index')} active={isActive('Index')}>
20               Accueil
21             </ResponsiveNavLink>
22             <ResponsiveNavLink className="nwhitec-red" href={route('About')} active={isActive('About')}>
23               À propos
24             </ResponsiveNavLink>
25             <ResponsiveNavLink className="nwhitec-red" href={route('Services')} active={isActive('Services')}>
26               Services
27             </ResponsiveNavLink>
28             <ResponsiveNavLink className="nwhitec-red" href={route('posts.index')} active={false}>
29               Publications
30             </ResponsiveNavLink>
31           </div>
32         </nav>
33       </header>
34     <main className="mx-auto max-w-5xl px-4 py-6">
+ 36       <Toast />
37       {children}
38     </main>
39   );
40 }
41
42 
```

```

30   public function share(Request $request): array
31   {
32     return array_merge(parent::share($request), [
33       'auth' => [
34         'user' => $request->user(),
35       ],
36       'flash' => [
37         'success' => session('success'),
38         'error'   => session('error'),
39       ],
40     ]);
41   }
42 }
43 
```

FIGURE 32
HandleInertiaRequest.php

FIGURE 33 – Layouts/AppLayout

Enfin, voici le résultat que vous devriez obtenir lorsque vous modifiez une publication.

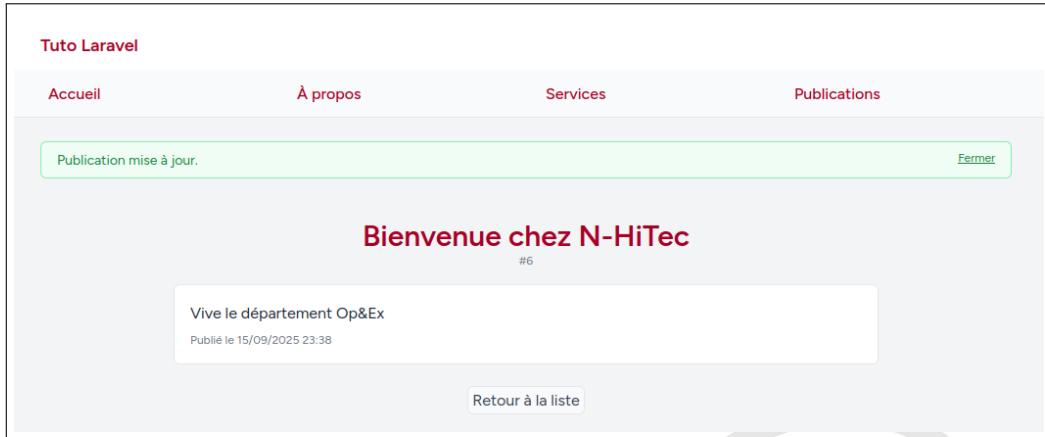


FIGURE 34 – Exemple du toast de modification

Vous pouvez avoir l'impression que nous en avons enfin fini avec cette formation, mais ... il reste quelque chose qui devrait vous chiffrer : N'importe qui peut écrire et modifier les publications.

36. Plus de détails sur les middlewares dans la section ??

Dans la Section ??, nous allons donc ajouter un système d'authentification très simple à notre site.

6 Authentication

6.1 Fortify

Pour cela, nous allons utiliser le package Fortify de Laravel .

“Laravel Fortify is a frontend agnostic authentication backend implementation for Laravel. Fortify registers the routes and controllers needed to implement all of Laravel’s authentication features, including login, registration, password reset, email verification, and more.” ([Doc Laravel](#) 

Cela veut donc dire qu’après configuration, nous n’aurons “plus qu’à” créer les views, et nous aurons finis ! Vous l’aurez deviné, la difficulté viendra donc de cette configuration.

6.1.1 Installation

L’installation est plutôt rapide, ces deux commandes suffisent :

1. sail composer require laravel/fortify
2. sail artisan vendor:publish --provider="Laravel\Fortify\FortifyServiceProvider"

Si vous êtes observateurs, vous aurez remarqué que cette dernière commande aura créé une nouvelle migration, qu’il faut donc déployer. Exécutez donc sail artisan migrate.

6.1.2 Configuration

Premièrement, ajoutez cette ligne dans config/app.php, dans la liste des providers utilisés par Laravel .

Ensuite, dans config/fortify.php, ajoutez /* */ pour commenter les éléments comme ci-dessous. L’array features permet de dire à Fortify quelles fonctionnalités vous souhaitez activer. En l’occurrence, nous sommes intéressés par registration() et resetPasswords(). Voici la liste des fonctionnalités :

- registration() permet l’authentification de base (register/login).
- resetPasswords() permet de réinitialiser son mot de passe.
- emailVerification() permet de vérifier les adresses emails des utilisateurs.
- updateProfileInformation() permet de modifier son profil.
- updatePassword() permet de modifier son mot de passe depuis son profil.

```

183 |     Illuminate\View\ViewServiceProvider::class,
184 |     App\Providers\FortifyServiceProvider::class,
185 |
186     'features' => [
187         Features::registration(),
188         Features::resetPasswords(), /*
189         // Features::emailVerification(),
190         Features::updateProfileInformation(),
191         Features::updatePasswords(),
192         Features::twoFactorAuthentication([
193             'confirm' => true,
194             'confirmPassword' => true,
195             // 'window' => 0,
196         ]),
197     ],

```

— twoFactorAuthentication permet d'activer la double authentification.

Ensuite, il faut dire à Fortify quelles `views` correspondent à quelles `routes`. Pour cela, ajoutez ceci dans `app/Providers/FortifyServiceProvider.php`, plus précisément dans la méthode `boot()` :

```
46     Fortify::loginView(function () {
47         return view('auth.login');
48     });
49
50     Fortify::registerView(function () {
51         return view('auth.register');
52     });
53
54     Fortify::requestPasswordResetLinkView(function () {
55         return view('auth.password.forgot');
56     });
57
58     Fortify::resetPasswordView(function () {
59         return view('auth.password.reset');
60     });
61 }
```

FIGURE 35

6.1.3 Views

Ensuite, il faut évidemment créer les `views` pour les fonctionnalités que nous gardons. Comme elles sont (très) longues et proviennent d'un package qui n'est plus officiellement supporté ([Laravel UI](#)), vous pouvez les trouver ici : <https://github.com/nhitec/Laravel-tutorial-auth-blades>. Placez le dossier `auth` dans `resources/views`, et le tour est joué !³⁷

Afin d'accéder à ces nouvelles pages, il faut que l'on modifie nos boutons dans `index.blade.php`. Nous allons ajouter les `routes` aux deux boutons existants, et en rajouter un troisième pour se déconnecter :

37. Remarquez que nous n'avons eu besoin ni de controllers ni de routes. C'est parce qu'ils sont déjà créés pour nous. Pour votre curiosité, vous pouvez trouver les routes créées par Fortify dans le fichier `vendor/laravel/fortify/routes/routes.php`.

Ceci est un document interne. Ne pas diffuser.

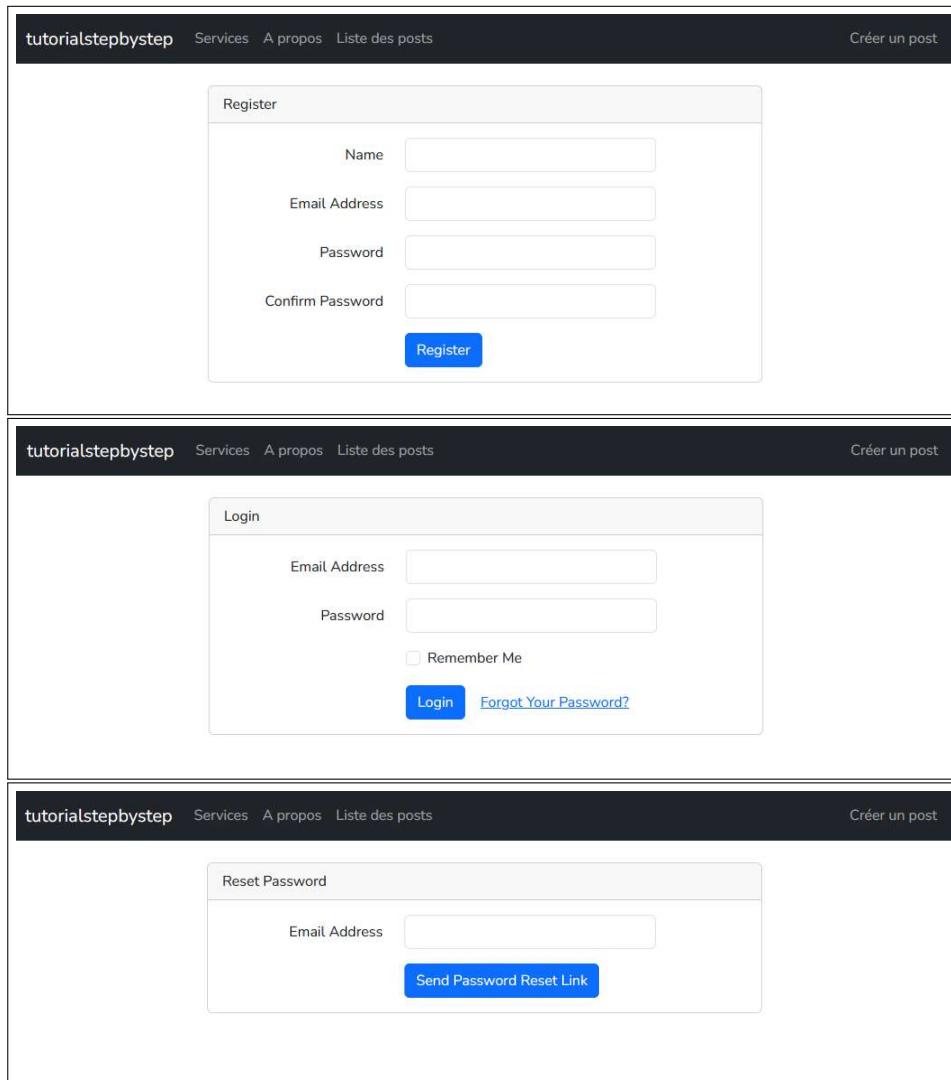
```
5  <div class="jumbotron text-center">
6    <h1> Accueil </h1>
7    <p> Ceci est le squelette de base pour le tutoriel d'introduction à la création de site web en utilisant laravel v10 ! </p>
8
9    <div class="d-inline-flex">
10      <div class="mx-1">
11        |   <a href="{{ route('login') }}" class="btn btn-lg btn-primary">Se connecter</a>
12      </div>
13      <div class="mx-1">
14        |   <a href="{{ route('register') }}" class="btn btn-lg btn-success">S'enregistrer</a>
15      </div>
16      <div class="mx-1">
17        <form action="{{ route('logout') }}" method="POST">
18          @csrf
19          <button href="{{ route('logout') }}" class="btn btn-lg btn-danger">Logout</button>
20        </form>
21      </div>
22    </div>
23
24  @endsection
```

FIGURE 36

Voici les pages que vous devriez obtenir :



Ceci est un document interne. Ne pas diffuser.



The figure consists of three vertically stacked screenshots of a web application interface. All three screenshots have a dark header bar with the text "tutorialstepbystep", "Services", "A propos", "Liste des posts", and "Créer un post".

- Screenshot (a): Register**
 This form is titled "Register". It contains four input fields: "Name", "Email Address", "Password", and "Confirm Password". Below the fields is a blue "Register" button.
- Screenshot (b): Login**
 This form is titled "Login". It contains two input fields: "Email Address" and "Password". Below the fields is a checkbox labeled "Remember Me". At the bottom are two buttons: a blue "Login" button and a blue link "Forgot Your Password?".
- Screenshot (c): Reset Password**
 This form is titled "Reset Password". It contains one input field: "Email Address". Below the field is a blue "Send Password Reset Link" button.

FIGURE 37 – Les quatres pages de gestions des posts.

Remarquez que le bouton de la FIGURE ?? ne fonctionne pas, nous réglerons ce problème dans une future Section (WIP).

6.2 Middlewares

Nous arrivons à la fin ! La dernière chose à faire, c'est protéger quelques routes. En effet, il est préférable que n'importe qui ne puisse pas accéder aux posts sans s'être préalablement authentifié. Il faut donc trouver un moyen de vérifier si un utilisateur est connecté avant de lui donner accès à la gestion/création de posts. C'est exactement à cela que servent les middlewares.

De manière générale, les middlewares sont des fonctions qui seront exécutées “entre deux requêtes”, pour par exemple interdire l'accès à une certaine route et rediriger l'utilisateur selon certaines conditions.

Les middlewares se trouvent dans le dossier app/Http/Middleware et sont configurés par le fichier app/Http/Kernel.php. Les middlewares que nous pouvons assigner manuellement aux routes/controllers sont ceux listés dans \$middlewareAliases.

En ce qui nous concerne, le middleware 'auth' est celui qu'il nous faut. Nous allons donc l'ajouter à aux routes responsables des posts dans routes/web.php comme cela :

```
23 | Route::resource('posts', PostsController::class)->middleware('auth');
```

FIGURE 38

Et voilà ! Désormais, lorsque vous essayez de voir ou de créer un post sans être connecté, vous devriez vous faire rediriger vers la page de connexion.



7 Conclusion

7.1 Postambule

Ce tutoriel touche (enfin) à sa fin ! Ce fût long et surement douloureux pour vous, mais vous y êtes arrivés, ce qui mérite déjà des félicitations en soi. Donc bravo !

Si vous avez toujours l'impression d'être un peu perdu, c'est tout à fait normal. Nous avons vu beaucoup de choses différentes, plusieurs langages différents, et des mécaniques qui s'entrecroisent. Ce n'est pas en suivant un tutoriel que vous deviendrez les maîtres de la programmation, et ce n'est pas l'objectif de ce tutoriel. Son objectif est de vous familiariser avec différentes notions afin de vous donner les clés nécessaires pour continuer de vous perfectionner et de comprendre les choses par vous mêmes. J'espère avoir accompli cet objectif !

7.2 Et ensuite ?

Il y a plusieurs suites possibles pour vous améliorer.

Premièrement, vous pouvez lire les documentations données au fil de tutoriel afin d'assimiler les notions et d'aller plus loin. Vous pouvez vous même essayer d'ajouter des fonctionnalités à ce mini-site (ou créer le vôtre de A à Z) pour tester vos nouvelles compétences.

Ou alors, si vous continuez l'aventure N-HiTec avec nous, vous aurez accès à encore plus de ressources pour continuer de vous former et d'apprendre des choses avec nous, et si vous le souhaitez, vous pourrez même ajouter votre pierre à l'édifice en 1) contribuant à cette formation, and 2) travaillant sur les nombreux projets que nous avons à proposer !

N-HiTec vous attend !



N-HiTec



Allée de la Découverte 10, 4000 Liège, Belgium
nhitec.com | info@nhitec.com