h e g

Haute école de gestion
Genève

Analyse du Framework NuxtJS pour le développement d'application web

Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES

par :

Evan SCHLERET

Conseiller au travail de Bachelor :

Rolf HAURI, Maître d'enseignement HES

Genève, le dimanche 30 juillet 2023

Haute École de Gestion de Genève (HEG-GE)

Filière Informatique de Gestion

Hes·so GENÈVE
Haute Ecole Spécialisée
de Suisse occidentale

Déclaration

Ce travail de Bachelor est réalisé dans le cadre de l'examen final de la Haute école de gestion de Genève, en vue de l'obtention du titre de Bachelor of Science HES-SO en Informatique de gestion.

L'étudiant a envoyé ce document par email à l'adresse remise par son conseiller au travail de Bachelor pour analyse par le logiciel de détection de plagiat URKUND, selon la procédure détaillée à l'URL suivante : https://www.urkund.com .

L'étudiant accepte, le cas échéant, la clause de confidentialité. L'utilisation des conclusions et recommandations formulées dans le travail de Bachelor, sans préjuger de leur valeur, n'engage ni la responsabilité de l'auteur, ni celle du conseiller au travail de Bachelor, du juré et de la HEG.

« J'atteste avoir réalisé seul le présent travail, sans avoir utilisé des sources autres que celles citées dans la bibliographie. »

Fait à Genève, le dimanche 30 juillet 2023

Evan Schleret

Remerciements

Je tiens ici à remercier sincèrement quelques personnes qui, par leurs actions concrètes, directement ou non liées à la rédaction de ce travail de Bachelor, l'ont rendu possible.

La première personne que je souhaite remercier est Gabriel Valero, mon meilleur ami, qui m'a vivement encouragé et poussé, il y quatre ans, à entreprendre ce Bachelor. Au-delà de cet aspect, je le remercie chaleureusement d'avoir toujours été présent, quelque puisse en être le contexte, pour rigoler, étudier, collaborer.

Je souhaite ensuite remercier plusieurs de mes autres amis, malheureusement sans pouvoir toutes et tous les citer : Raphaël Vidreiro, Quentin Gonzalez, Bruno Mota Cardoso, Matias Salgueiro, Ana Moreira, Mathias Rullo, Augustin Hans, Matei Ionescu et Daniel Lemos Coutinho, qui ont tous apporté des briques importantes dans les fondations de ma vie. Une attention particulière à mon ami Alexandre Favre qui a accepté de relire mon travail de Bachelor plusieurs fois et qui à chaque fois m'a apporté de précieux conseils dans la façon dont je pouvais présenter certains concepts.

Je remercie également mes deux parents, qui m'ont toujours soutenu dans mes choix et décisions, et qui m'ont toujours apporté une partie du support nécessaire à mes accomplissements. Je remercie également ma grand-mère et mon beau-grand-père, qui m'ont eux aussi toujours soutenu. Pour terminer avec la famille, je remercie ma sœur, Anastasia Schleret, pour sa présence et son soutien.

Je témoigne également une vive reconnaissance et des remerciements sincères à mon parrain, Frédéric Gisiger, qui au cours de ces derniers mois, m'a appris à m'améliorer sur de nombreux aspects.

Je suis reconnaissant envers mon directeur de Bachelor, Rolf Hauri, qui a accepté de me suivre, qui m'a donné de son temps et qui m'a prodigué de nombreux conseils très utiles.

J'exprime également mes remerciements à tous les membres de ma famille, mes amis et mes proches que je n'ai pas directement nommés, qui ont contribué à rendre possible ce Bachelor.

Résumé

Mon travail porte sur la version 3 de Nuxt.js, un framework JavaScript sorti au début de l'année 2023, se plaçant comme un sérieux concurrent par rapport à nombreux de ses homologues.

La problématique que je pose pour ce travail de Bachelor gravite autour d'un certain nombre de questions similaires : comment Nuxt se compare-t-il, tous points de vue confondus, des frameworks comparables ? Comment Nuxt peut être plus performant que sa version majeure précédente ou à l'égard des autres frameworks ? Comment Nuxt peut être un grand allié des développeurs lors de la création et la conception de leurs applications web ? Enfin, comment Nuxt peut-il être utilisé pour développer des applications respectueuses des standards modernes en matière de SEO ?

Je vais vous montrer que Nuxt est un framework très polyvalent et répond à beaucoup des besoins rencontrés dans le développement d'applications web : les performances délivrées, la façon dont il peut facilement être optimisé pour le référencement naturel par les moteurs de recherche, comment il permet de communiquer avec des APIs ou mettre en cache le contenu que ces dernières retournent. Je vous montrerai également comment Nuxt peut vous faire gagner un temps considérable lors du développement de vos applications, avec l'ensemble des fonctionnalités qu'il propose « out-of-the-box », donc sans configuration particulière supplémentaire.

Afin de transformer la théorie que je vous présenterai tout au long de ce travail de Bachelor, je finirai par développer une application avec Nuxt en essayant d'inclure le maximum de fonctionnalités que je vous aurai présentées, en expliquant mon cheminement de pensée.

Table des matières

Déclaration i
Remerciements ii
Résumé iii
Liste des tableaux vi
Liste des figures vi
Liste des acronymes vii
1. Introduction 1
1.1 Contexte du travail de Bachelor 1
1.2 La problématique 1
1.3 Les objectifs de recherche 1
2. Exploration des frameworks et bibliothèques de développement web 2
2.1 Introduction aux frameworks et bibliothèques de développement web .. 2
2.2 Comparaison de frameworks web 4
2.2.1 Les frameworks JavaScript, leurs atouts et différences 4
2.2.1.1 Les architectures d'application 5
2.2.1.2 Les approches de développement 6
2.3 TypeScript vs JavaScript 7
2.3.1 Le typage statique 7
2.3.2 Les annotations de type 8
2.3.3 Le support avancé d'ECMAScript 9
2.3.4 Les inférences de type 10
2.4 Nuxt 11
3. Problèmes et besoins en matière de développement web : comment Nuxt peut vous aider à résoudre certaines situations 12
3.1 Mise en contexte et pose des problématiques 12
3.2 Comment Nuxt peut-il être un bon candidat pour résoudre, répondre ou améliorer ces situations ? 14
3.2.1 L'expérience de développement et la préparation de l'application pour la mise en production 14
3.2.2 L'expérience utilisateur 15
3.2.3 Les performances de son moteur de serveur Nitro 15
3.2.4 La maintenabilité et scalabilité 16
3.2.5 L'aspect souvent négligé : la sécurité 16
4. Présentation approfondie de Nuxt 3 18
4.1 Les fonctionnalités apportées par dans Nuxt 18
4.1.1 Le Hot module reloading (ou rechargement de module à chaud) 18
4.1.2 Le server-side rendering (ou SSR) 19
4.1.3 Le déploiement « on the edge » (déploitement CDN) 21

4.1.4 L'automatisations du framework et ses conventions....21
4.1.4.1 Génération automatique de vos routes....21
4.1.4.2 Auto-importation des composants....23
4.1.5 Les façons de récupérer des données sur des API....24
4.1.6 Nitro : le nouveau moteur de serveur de Nuxt....25
4.2 Analyse des différences avec la version majeure précédente (Nuxt 2)...26
5. Conception de l'application....28
5.1 Description de l'architecture de mon application....28
5.2 Justification des choix technologiques....29
5.2.1 L'ecosystème de Nuxt au travers de celui de Vue....30
5.2.2 Le SSR....30
5.2.3 Le système d'auto-importation....31
5.2.4 Le routage dynamique....31
5.2.5 L'intégration de bibliothèques....32
6. Évaluation des performances de Nuxt....35
7. Développement de l'application....39
7.1 Comment créer un projet Nuxt....39
7.2 Concept et fonctionnalités de l'application....39
7.3 Commentaires et explications sur certaines parties du développement et du code de l'application....40
Conclusion....54
Bibliographie....58

Liste des tableaux

Tableau 1 : analyse des fonctionnalités et des approches de quatre frameworks JavaScript 5

Liste des figures

Figure 1 : exemple de type dynamique avec JavaScript qui ne fonctionnerait pas avec TypeScript à typage statique 8
Figure 2 : exemple d'annotation de type de variables en TypeScript 8
Figure 3 : exemple d'annotation de types pour un objet en TypeScript 9
Figure 4 : exemple d'annotation de type pour les paramètres d'une fonction en TypeScript 9
Figure 5 : exemple d'inférence de type de variable en TypeScript 10
Figure 6 : exemple simple de structure de pages d'une application Nuxt 22
Figure 7 : exemple de structure de page avec un paramètre dynamique dans une application Nuxt 23
Figure 8 : tableau comparatif des différences notables entre Nuxt 2 et 3 27
Figure 9 : exemple de composant Nuxt 28
Figure 10 : exemple de page intégrant plusieurs fois un même composant Nuxt 28
Figure 11 : représentation de l'architecture de mon application Nuxt 29
Figure 12 : schéma de comparaison sur l'installation de tailwindcss sur le framework Next.js et Nuxt 33
Figure 28 : comparatif entre Nuxt et Next en nombre de requêtes par seconde 35
Figure 29 : temps moyen de réponse entre Nuxt et Next 36
Figure 30 : routes créées pour le benchmark d'icarusgk 36
Figure 31 : comparatif sur le nombre de requêtes par secondes absorbées par Nuxt et Next en fonction du nombre de tags affichés 37
Figure 32 : comparatif sur le temps moyen de réponse entre Nuxt et Next en fonction du nombre de tags affichés 37
Figure 33 : comparatif du nombre de requêtes gérées par Nuxt et Next sur le Hello World 38
Figure 13 : commande pour installer un projet Nuxt 39
Figure 14 : importation d'un composant fait pour Vue dans une application Nuxt 43
Figure 15 : appel du composant dans une page de l'applciation 43
Figure 16 : composable personnalisé useApiFetch 44
Figure 17 : exemple d'utilisation de la métohode useDateFormat 45
Figure 18: affichage conditionnel du bouton d'édition du billet de blog 46
Figure 19 : sections des commentaires avec l'affichage conditionnel du bouton « supprimer » 46
Figure 20 : exemple de composant envoyant un évènement à son parent 47
Figure 21 : composant Comment de mon application 48
Figure 22 : intégration du composant Comment dans une page 49
Figure 23 : affichage de la page d'accueil et de ses filtres 49
Figure 24 : affichage des requêtes effectuées quand on met à jour un composant réactif de Vue et que useFetch le surveille 50
Figure 25 : page d'accueil avec les filtres 51
Figure 26 : middleware permettant de vérifier si l'auteur du blog est le même que celui qui est connecté 52
Figure 27 : adaptation de la page de création de compte pour prendre en compte l'ajout d'une image de profil 53

Liste des acronymes

Acronyme	Désignation
Nuxt	Désigne le framework Nuxt. À moins que cela soit précisé explicitement, ne fait référence qu'à la version 3 du framework.
Vue	Désigne le framework Vue.js. À moins que cela soit précisé explicitement, ne fait référence qu'à la version 3 du framework.
JSON	JavaScript Object Notation. Format de données.
ECMAScript	European Computer Manufacturers Associate Script. Un langage de scripting basé sur JavaScript.
SSR	Server side rendering
PHP	Acronyme récursif qui signifie PHP Hypertext Preprocessor.

1. Introduction

1.1 Contexte du travail de Bachelor

Parmi l'ensemble des cours que j'ai pu étudier à la Haute École de Gestion de Genève lors de mon Bachelor d'informaticien de gestion, il fallait que j'en choisisse un seul, celui dont j'aurai le plus de plaisir à parler et présenter. J'ai donc choisi le vaste domaine du développement web.

L'écriture de mon Bachelor coïncide avec quelques semaines d'écart avec le lancement d'une nouvelle version d'un framework que je souhaitais apprendre : Nuxt.js, que je désignerai toujours ci-après par le terme « Nuxt », qui fera appel à sa version 3, sauf si je précise explicitement une autre version.

1.2 La problématique

La problématique de ce travail de Bachelor gravite autour d'un certain nombre de questions similaires : comment Nuxt peut-il être plus performant que sa version majeure précédente (la version 2.x) et quelles sont ses améliorations ? Comment Nuxt se compare-t-il aux autres frameworks comparables ? Comment Nuxt pourrait-il aider un développeur, ou une équipe de développeurs, à créer des applications web, tout en garantissant un travail évolutif et performant ? Enfin, pour ne formuler qu'une quatrième question similaire : comment Nuxt peut-il être utilisé pour développer des applications web respectueuses des standards et bonnes pratiques en matière de SEO ?

Ce sont les questions auxquelles je vais tenter le plus fidèlement possible de répondre, en procédant à l'analyse du framework Nuxt pour le développement d'applications web.

1.3 Les objectifs de recherche

L'objectif de ce travail de Bachelor est simple : analyser et expliquer pourquoi Nuxt est un framework intéressant, novateur et utile dans la démarche de création d'une application web, que vous soyez un développeur qui travaille en freelance ou que vous soyez une grande entreprise active dans le développement web.

Nuxt est un framework très en vogue et qui prend de plus en plus en popularité. Basé sur le framework Vue.js 3 – ci-après toujours désigné « Vue » – Nuxt vient ajouter des fonctionnalités internes très intéressantes ayant pour but de faciliter et optimiser la façon dont le développement de ce type d'application web est fait.

2. Exploration des frameworks et bibliothèques de développement web

Dans ce chapitre, nous aborderons le vaste sujet que représente l'univers des frameworks et bibliothèques spécialisées pour le développement d'applications web.

Je ferai d'abord une petite introduction sur le sujet, pour expliquer de quoi il s'agit, puis je présenterai une comparaison entre quelques frameworks. Je vous parlerai ensuite des différences entre JavaScript et TypeScript avant de présenter très brièvement Nuxt, la partie plus précise arrivant plus tard, au chapitre 4.

2.1 Introduction aux frameworks et bibliothèques de développement web

Il existe aujourd'hui pléthores de frameworks et de bibliothèques utilisées dans le cadre de développement d'applications web. D'abord, il convient de poser la question suivante : qu'est-ce qu'un framework ou une bibliothèque logicielle ?

« Un framework Web ou framework d'application Web est un framework logiciel conçu pour prendre en charge le développement d'applications Web, notamment des services Web, des ressources Web et des API Web. Les frameworks Web fournissent un moyen standard de créer et de déployer des applications Web sur le World Wide Web. Les frameworks Web visent à automatiser les mécanismes les plus courants du développement Web. Par exemple, de nombreux frameworks Web fournissent des bibliothèques pour l'accès aux bases de données, les moteurs de rendu et la gestion des sessions, et ils favorisent souvent la réutilisation du code. »[1] - Wikipédia

Comme expliqué ci-dessus, on peut résumer que dans le contexte de programmation en informatique, un framework va être un ensemble d'outils, de composants, de méthodes ou de fonctions, qui a pour but de créer un cadre et un socle pour le développement d'un logiciel.

Dans la situation d'une application web, un framework est semblable à la description faite, à ceci près qu'on y trouve donc plus souvent diverses API, ressources, modules de gestion de l'authentification, de la base de données, etc. centrés autour d'une utilisation faite pour le web, accessible d'abord aux navigateurs web.

On peut représenter un framework par l'analogie suivante : un framework est comme une grande boîte de briques de Lego. L'aspect pratique de la chose est que nous n'avons donc pas besoin de dessiner, mouler et industrialiser tous les différents types

[1] https://fr.wikipedia.org/wiki/Framework_Web

de briques. On peut alors les utiliser plus ou moins aisément pour parvenir à un but donné : construire une application web, en ayant par exemple la possibilité d'implémenter facilement des standards en matière de SEO ou de sécurité. Le framework va alors agir en un sens pratique pour tout développeur web : faciliter son travail en lui permettant de ne pas se soucier de toutes les parties souvent longues et compliquées à mettre en place et à sécuriser. Là où il perdrait en temps à implémenter des concepts connus, il en gagne maintenant utilement pour ses missions et projets.

Prenons ensemble un exemple facile à illustrer qui est lié à une partie importante des applications web dites dynamiques : la gestion de la session. La gestion de session permet aux utilisateurs d'une application web d'être identifiés et de pouvoir rester connectés en ayant une expérience unique ; non pas par la qualité de sa visite sur l'application mais bien par les fonctionnalités qu'il utilisera. Par exemple sur un forum, il pourra envoyer ses propres messages et interagir avec ceux des autres. La gestion de la session d'un utilisateur implique d'autres aspects importants comme la gestion des cookies, la protection contre les attaques de type CSRF, par force brute ou encore les injections SQL. L'authentification fait également partie intégrante de la gestion de la session. Elle est le douanier de l'application, celle qui permet de donner les autorisations que l'utilisateur doit ou non avoir sur l'application. S'assurer de l'identité numérique d'un utilisateur est primordial pour garantir que les accès donnés à une ressource soient correctement donnés à la bonne personne. Vous l'aurez compris, le framework va faciliter le travail du développeur en s'occupant, pour l'exemple traité ici, de la gestion de la sécurité et colmater diverses failles de sécurité connues et non résolues.

Parlons maintenant de la bibliothèque logicielle, ci-après désignée par le simple terme bibliothèque. La bibliothèque est comme le sniper d'une partie d'un framework. Sa mission est de fournir un ensemble de méthodes bien précis ne servant qu'à un but bien déterminé et duquel elle ne sort en principe pas ; raison pour laquelle il existe logiquement plus de bibliothèques que de frameworks. La bibliothèque est utilisée comme support d'un framework, au travers de l'utilité qu'elle a. Par exemple, la bibliothèque Faker[2] est utilisée dans un strict but de générer de fausses données, à priori plus souvent utilisée pour remplir artificiellement les données d'une base de données à des fins de tests. L'avantage de la bibliothèque est qu'elle est utilisable dans tous les projets qui s'y prêtent. Elle est mobile et flexible aux projets dans lesquels on l'intègre.

Ainsi, on peut dire que le framework est à la bibliothèque ce qu'une recette de cuisine est à ses ingrédients : la recette raconte comment et quand utiliser les ingrédients, mais

[2] https://fakerjs.dev/

elle laisse la place à l'improvisation et aux essais, dans le but de créer un plat savoureux et intéressant à la personne qui le dégustera. Le framework offre au développeur une structure de base, un ensemble de recettes pour chaque ingrédients – les bibliothèques – qu'il pourra intégrer comme il le souhaite, tenant compte de standards, normes, etc. Les ingrédients de la recette sont les bibliothèques : flexibles et utilisables dans différentes recettes de cuisine ou frameworks.

2.2 Comparaison de frameworks web

Pour comparer différents frameworks web, il faut d'abord se poser la question suivante : en quels langages peut-on écrire ce type de programme[3]. La réponse, qui n'en est donc pas vraiment une est simple : il n'y a pas de vérité claire et exhaustive car il est possible de développer un framework avec presque n'importe quel langage.

En revanche, il est correct d'affirmer, en regardant une partie des frameworks disponibles parmi les plus populaires aujourd'hui, que certains langages sont plus souvent utilisés que d'autres car ils répondent peut-être mieux à un besoin recherché, selon les fonctionnalités, l'approche souhaitée ou la facilité d'usage qu'ils ont à offrir dans ce type de développement.

On retrouve alors par exemple les frameworks JavaScript avec React, Angular, Vue.js, Nuxt.js ou encore Next.js, le PHP avec Laravel, Symfony ou CakePHP, le Python qui propose Django et Flask notamment, Ruby avec Ruby on Rails, Sinatra ou encore même Java qui propose notamment Spring, pour ne citer que lui. Il est également possible d'utiliser le C#, avec le développement d'applications de type ASP.NET.

Il est important de noter que tous ces frameworks n'ont pas la même approche de base. Certains ont une architecture dite MVC pour Modèle-Vue-Contrôleur[4] : Laravel, Symfony, CakePHP ou encore Ruby on Rails proposent cette approche, qui se distingue par exemple des frameworks JavaScript. Vue.js est par exemple de type MVVM pour Modèle-Vue-VueModèle[5]. Nuxt, basé pour rappel sur Vue, suit les mêmes prémices, tout comme Angular.

2.2.1 Les frameworks JavaScript, leurs atouts et différences

Penchons-nous sur l'utilisation du langage JavaScript pour le développement de frameworks. Je vais comparer quatre des nombreux frameworks disponibles dans ce langage : Vue.js, Nuxt.js, Angular et React.

[3] https://captiva-it.com/blog/tribu-open-source/langages-de-programmation-et-framework
[4] https://fr.wikipedia.org/w/index.php?title=Modèle-vue-contrôleur&oldid=205208224
[5] https://en.wikipedia.org/w/index.php?title=Model–view–viewmodel&oldid=1164384181

Tableau 1 : analyse des fonctionnalités et des approches de quatre frameworks JavaScript

	Vue.js	Nuxt.js	Angular	React
Approche de développement	Déclarative	Déclarative	Impérative	Déclarative
Communauté	Grande	Moyenne mais grandissante	Grande	Grande
Architecture	Composant	Composant	MVC	Composant
Intégration d'outils et bibliothèques	Facile	Facile	Moyenne	Facile

(Evan Schleret)

Un point commun que nous trouvons entre ces frameworks est l'architecture en composants.

Pour information, si vous vous référez au tableau, vous observerez que les notions de performances et de courbe d'apprentissage ne sont pas présentes car elles peuvent être perçues comme subjectives car trop le reflet du sentiment de tout un chacun et des avis des différentes communautés.

2.2.1.1 Les architectures d'application

L'architecture en composants est un mode de développement consistant à diviser une application – ici une application JavaScript – en un ensemble de plus petits éléments autonomes les uns des autres, qu'on appelle composants.[6] Chacun d'eux est responsable de lui-même et d'une tâche spécifique qui pourra être réutilisée dans d'autres parties de l'application (repensez à mon analogie des briques de Lego).

Imaginez par exemple un tableau énumérant une liste d'utilisateurs. Conceptualisez maintenant le fait que chaque ligne de ce tableau soit un composant qui permette l'édition des informations de l'utilisateur. Le composant qu'on pourrait appeler « ligne de tableau » aura toute la logique applicative permettant l'édition et l'affichage de cette

[6] https://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_composant

même ligne. On y trouvera alors toutes les propriétés et méthodes permettant de fonctionner, permettant d'alterner entre visualisation ou édition des données, l'enregistrement et la validation desdites données saisies.

Bien que ces composants soient censés être indépendants les uns des autres, il peut parfois être pratique de les faire communiquer ensemble. Pour combler ce besoin, nous pouvons émettre des informations depuis un composant et écouter les événements dans d'autres, comme cela se fait par exemple via les *Component Events*[7] de Vue. On peut également imbriquer ces mêmes composants les uns dans les autres, créant alors une notion de hiérarchie de composants. Par exemple, le composant « barre de navigation » contiendra un autre composant « élément de barre de navigation », qui pourrait lui aussi contenir un composant « lien d'élément de barre de navigation ».

Pour résumer ce qui précède, l'architecture présentée ici, l'architecture en composants, est l'inverse d'une architecture plus standard comme le modèle MVC. Les composants permettent une plus grande modularité et une bien meilleure réutilisation du code. Cela permettra au développeur de s'assurer de ne pas avoir à modifier n fois une portion de code car elle serait présente n fois sur son application. Il peut également s'assurer que la maintenance et l'évolution pourra lui être plus facile, ce qui ne manquera naturellement pas de lui faire plaisir à lui comme à tous les développeurs qui pourraient passer derrière lui en contribuant au développement de l'application. Enfin, cette architecture permet également une meilleure lisibilité du code car comme je le disais, on peut séparer chaque tâche, action ou fonctionnalité en fonction du but précis pour laquelle elle est pensée et ce à quoi elle doit servir.

2.2.1.2 Les approches de développement

Quand on veut expliquer la différence entre l'approche de développement dite déclarative ou impérative, il faut surtout comprendre qu'il s'agit d'une question de paradigme : l'approche déclarative décrit toujours le résultat souhaité[8], l'objectif. À contrario, l'approche impérative décrira étape par étape ce qui se passera[9], à la façon dont le ferait par exemple un script. Des deux méthodes, c'est elle qui existe depuis le plus longtemps.

Schématisons et comparons davantage les deux méthodes. Imaginez que l'approche impérative soit une liste détaillée qui contienne toutes les étapes nécessaires pour vous

[7] https://vuejs.org/guide/components/events.html
[8] https://www.ionos.fr/digitalguide/sites-internet/developpement-web/programmation-declarative/
[9] https://fr.wikipedia.org/wiki/Programmation_imp%C3%A9rative

déplacer d'un point A à un point B. Pour remplir ce cahier des charges, vous devrez par exemple tourner à gauche au prochain feu, rouler cent cinquante mètres, tourner à droite, etc.

L'approche déclarative peut quant à elle être imaginée comme une représentation d'une carte montrant le chemin le plus direct, le plus court ou le plus économique, de ce même point trajet. Au lieu de suivre scrupuleusement une suite d'instructions tel que vous l'auriez fait dans l'exemple précédent, vous n'auriez qu'à suivre un tracé donné et prendre les bonnes décisions pour atteindre votre destination.

On peut retenir que l'approche impérative est parfois plus efficace, mais est parfois surtout beaucoup plus fastidieuse et sujette à des erreurs. L'approche déclarative est souvent plus facile à comprendre mais elle nécessitera que vous ayez une bonne compréhension générale de votre environnement et de votre carte, pour reprendre notre analogie précédente.

Ainsi, dans le cadre du développement d'une application web, une approche impérative nécessitera que vous ayez une attention particulièrement détaillée à chaque étape du processus d'un composant ou d'une méthode, alors qu'une approche déclarative vous permettrait de davantage vous concentrer sur l'objectif final que vous souhaitez atteindre, en vous laissant écrire des étapes bien plus succinctes, plus simples et davantage lisibles.

2.3 TypeScript vs JavaScript

Si la première version de TypeScript n'est sortie que le 9 février 2012, JavaScript est bien plus vieux, sorti pour sa part au mois de mai 1996.

Il faut aujourd'hui voir TypeScript comme un sur-ensemble de JavaScript. En effet, tout le code que vous pourriez produire avec JavaScript le sera également avec TypeScript.

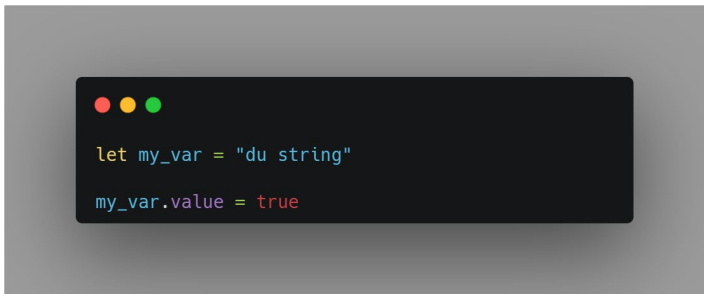
Nous pouvons néanmoins étudier la différence entre ces deux langages, bien qu'ayant donc un tronc commun, selon les aspects suivants : le typage statique, les annotations de type, le support pour les fonctionnalités ECMAScript les plus récentes et les inférences de type.

2.3.1 Le typage statique

Le TypeScript est un langage à typage dit « statique ». Cela signifie que l'ensemble des types de variables que vous pourrez utiliser seront vérifiés à la compilation de votre code, évitant alors par avance des éventuels problèmes.

JavaScript a un typage dit dynamique et permet donc par exemple de faire la chose ci-dessous, qui ne fonctionnerait pas avec TypeScript.

Figure 1 : exemple de type dynamique avec JavaScript qui ne fonctionnerait pas avec TypeScript à typage statique



(Evan Schleret)

Comme je le disais juste avant, l'exemple ci-dessus avec TypeScript provoquerait une erreur qui indiquerai qu'une valeur booléenne n'est pas assignable à une variable de type string.

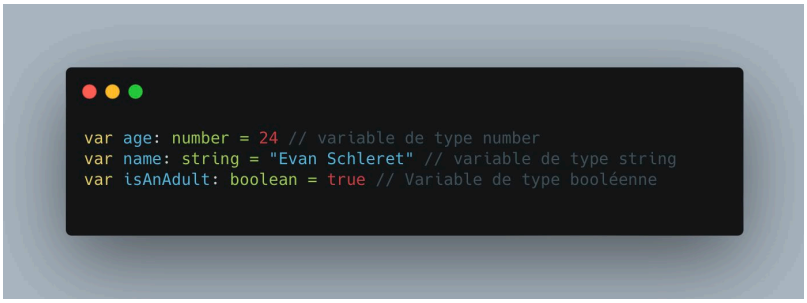
Outre la détection en amont des erreurs de typage, l'utilisation de TypeScript présente des avantages majeurs en matière de maintenance du code. Nous en parlerons au point suivant mais les annotations de type fournissent une documentation on ne peut plus explicite sur les différents types de variables attendus.

Avec TypeScript, l'auto-complétion est par ailleurs améliorées pour les IDE qui la supporte, car l'IDE sait exactement ce avec quoi il travaille. Cela peut avoir pour effet de rendre le développement plus fluide et peut réduire les erreurs liées à des problèmes de typage de variable.

2.3.2 Les annotations de type

Les annotations de type servent à définir le type des variables que vous utilisez : string, booléen, integer, etc. Voyez par vous-même avec l'exemple ci-dessous.

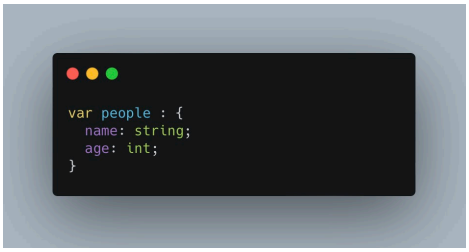
Figure 2 : exemple d'annotation de type de variables en TypeScript



(Evan Schleret)

Il est utile de savoir que vous pouvez également faire cette annotation de type sur les fonctions et les objets.

Figure 3 : exemple d'annotation de types pour un objet en TypeScript



(Evan Schleret)

Figure 4 : exemple d'annotation de type pour les paramètres d'une fonction en TypeScript

(Evan Schleret)

Le fait de déclarer les types de variables, fonctions et objets permet comme je le disais au point précédent de réduire les erreurs car tout est vérifié lors de la compilation de votre code. Les IDE modernes iront même vérifier ça avant, en procédant à des pré-compilations, en vous disant par exemple directement qu'il n'est pas possible d'assigner « John » à la variable « age » de l'objet « people » de la figure 3.

L'assignation de type n'est pas possible en faisant du pur JavaScript.

2.3.3 Le support avancé d'ECMAScript

ECMAScript est le support du standard sur lequel JavaScript a été conçu et construit[10]. C'est sur lui que tout repose. TypeScript, comme je le disais, est un sur-ensemble – une surcouche, pourrait-on dire – de JavaScript. Il utilise donc la même base que JavaScript, en se reposant lui aussi sur ce standard, en étendant ses fonctionnalités pour permettre de nouvelles possibilités et mécanismes que JavaScript ne fournit pas nativement.

L'avantage que TypeScript fournit avec le support d'ECMAScript est que n'importe quel vieux code JavaScript fonctionnera dans un fichier TypeScript et ce, sans devoir le changer. Ça permet de pouvoir garder du vieux code d'offrir la possibilité de le

[10] https://fr.wikipedia.org/wiki/ECMAScript

refactoriser et le modifier en le transformant en TypeScript pour pouvoir profiter à 100% de l'ensemble de ses avantages.

TypeScript introduit également de nouvelles fonctionnalités qu'on ne trouverait pas nativement avec JavaScript. On peut notamment utiliser les concepts tels que les interfaces, les énumérations, les types génériques ou les modules.

Quelque part, TypeScript donne une dimension beaucoup plus tournée vers ce qui pourrait ressembler à de l'orienté objet.

2.3.4 Les inférences de type

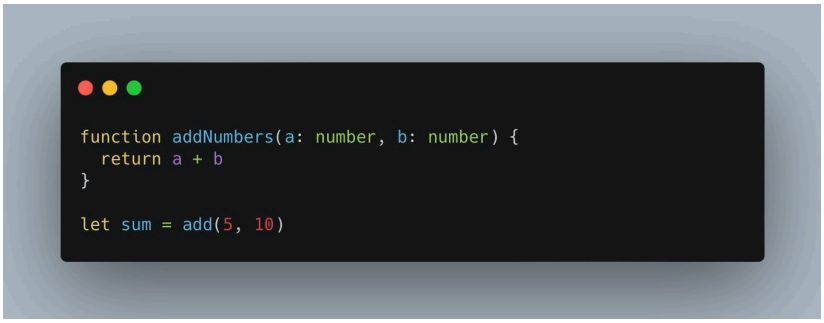
Les inférences de type interviennent quand vous ne spécifiez justement pas les types de vos variables, les types retournées par vos fonctions, etc[11].

Cela signifie que TypeScript va automatiquement, et de façon autonome, détecter le type le plus probable de la variable que vous déclarez et la lui assignera ensuite. Le type de la variable est donc influencé par la valeur que vous lui définissez.

Ensuite, tout se passe comme le ferait TypeScript. Le type une fois affecté à une variable, ne vous permettra plus de lui assigner n'importe quoi d'autre que ce qu'elle est censée accepter. Si vous essayez quand même en tentant par exemple d'assigner un nombre entier (un integer) à une variable afférée comme étant une chaîne de caractères (un string), le compilateur vous bloquera en vous signifiant que vous ne pouvez pas affecter un integer à un string.

TypeScript peut le faire de façon plus complexe, en inférant une variable en fonction du résultat retourné par une fonction.

Figure 5 : exemple d'inférence de type de variable en TypeScript



(Evan Schleret)

[11] https://fr.wikipedia.org/wiki/Inf%C3%A9rence_de_types

Dans l'exemple ci-dessus, la variable « sum » est afférée par TypeScript comme étant de type « number ».

2.4 Nuxt

Pour ne présenter que très brièvement Nuxt, car j'en parlerai bien plus en détail dans les prochaines parties de ce travail de Bachelor, il s'agit donc d'un framework gratuit et open-source. Comme les créateurs de Nuxt le décrivent, il permet « de façon intuitive et extensible de créer des application web et des sites web complets et sécurisés, performants et de qualité, prêts aux environnements de production, le tout avec Vue.js » [12]. La version 3 de ce framework, celle que je traite dans le cadre de ce travail de Bachelor, a été basée sur Vue 3. Cette version du framework est sortie au mois de janvier 2023.

[12] https://nuxt.com/docs/getting-started/introduction

3. Problèmes et besoins en matière de développement web : comment Nuxt peut vous aider à résoudre certaines situations

3.1 Mise en contexte et pose des problématiques

Chaque développeur, qu'il travaille seul ou en équipe mais qu'il souhaite développer une application web s'est normalement posé plusieurs questions avant de se lancer tête baissée (il faut tout du moins l'espérer pour lui). Il n'y a pas forcément de réponse complètement juste ou fausse, mais il faut garder à l'esprit que chaque choix amène son lot de conséquences à plus ou moins long terme et qu'il est donc judicieux de se poser et réfléchir convenablement au choix qui sera fait.

Permettons-nous de prendre quelques exemples de problématiques communes à des projets de développement web. Je précise qu'il s'agit bien sûr d'une liste non exhaustive.

Le premier problème qu'un développeur pourrait résoudre est bien sûr le choix de la technologie qu'il va adopter pour son projet. En effet, sélectionner choisir la bonne technologie – le bon framework – pour son application est essentiel. Chaque option lui offrira un vaste éventail de possibilités et fonctionnalités. Ce choix devra bien sûr être déterminé par les besoins du projet et l'utilisation de l'application dans le futur. Il conviendra également de devoir adapter le choix arrêté en fonction des contraintes de sécurité imposées, des performances attendues ou encore de la scalabilité requise.

Un second problème réside dans la conception et l'architecture de l'application. Il est effectivement nécessaire, pour éviter de perdre du temps, que cette décision ne soit pas prise à la légère et qu'elle soit faite correctement pour garantir une maintenabilité, de la flexibilité, mais également l'extensibilité de son application. Le choix qui sera fait est primordial car c'est dès le début du développement qu'il convient d'adopter une base solide en termes de conception, d'organisation du code ou encore de la planification de la structure d'une éventuelle base de données. La refactorisation et la réécriture complète ou partielle prennent du temps. Il faut alors correctement choisir son architecture pour s'assurer que le crédit de temps qu'on va allouer à un projet est utilisé de façon utile, efficace et intelligente. Quelque soit le contexte, le temps perdu est souvent compliqué à rattraper.

L'expérience utilisateur et la simplicité de navigation dans l'application que vous serez amené à développer est également importante. Au-delà de savoir comment concevoir des interfaces utilisateurs, il convient de rappeler que la simplicité d'une telle interface

ne se traduit pas toujours par une absence de complexité. Vous situez la nuance ? Une interface peut être simple en apparence mais complexe par son usage.

Le site User Inyerface[13] explique tout à fait ce concept en exagérant ce à quoi une interface web ne doit absolument pas ressembler.

Sir Jonathan Ive, alors en charge de l'équipe du design des produits chez Apple entre 1996 et 2019[14], disait « Simplicity is not the absence of clutter, that's a consequence of simplicity. Simplicity is somehow essentially describing the purpose and place of an object and product. The absence of clutter is just a clutter-free product. That's not simple. »[15] - Sir Jonathan Ive.

Cette citation exprime bien toute l'essence et l'idée même de la conception d'une bonne interface utilisateur : la simplicité d'une interface utilisateur est bien davantage qu'une simple absence de désordre ou de complexité. Elle implique plutôt une conception réfléchie en amont qui définira clairement l'objectif et la place des éléments graphiques. Cette étape est d'autant plus importante qu'elle pourrait vous demander beaucoup de temps si elle est bâclée ou si des changements majeurs devaient être nécessaires par la suite.

La question des performances est aussi très importante. Votre application accueillera vraisemblablement des utilisateurs (je vous le souhaite en tout cas !) qui ne souhaiteront certainement pas attendre plusieurs secondes pour un chargement de page, pouvant créer un sentiment de frustration face aux quelques secondes qu'on semblera avoir perdu, car alors purement investies à devoir attendre. Votre application devra donc être optimisée de sorte à répondre efficacement et rapidement aux requêtes de vos utilisateurs. Il sera attendu de votre application un temps de chargement aussi bas que possible. Il faudra donc optimiser d'éventuelles requêtes de base de données, faire de la mise en cache, compresser des données transmises à l'utilisateur tel que la stylisation de l'application et le JavaScript.

Je l'évoquais également précédemment, les problématiques liées à la maintenabilité et la scalabilité sont importantes. Une fois que votre application sera mise en production, il serait curieux de la considérer comme un produit fini qui ne sera plus mis à jour ou qui ne subira aucune évolution quelconque. De ce fait, il est important de choisir la

[13] https://userinyerface.com/
[14] https://fr.wikipedia.org/w/index.php?title=Jonathan_Ive&oldid=205506990
[15] https://www.telegraph.co.uk/technology/apple/9283706/Jonathan-Ive-interview-simplicity-isnt-simple.html

technologie, le framework que vous allez utiliser, de façon à vous laisser toute la liberté de le mettre à jour et le maintenir facilement.

Enfin, le dernier problème, mais de loin pas des moindres ou des moins importants, est l'aspect de la sécurité, qu'il ne faut à aucun prix négliger. C'est un aspect malheureusement pas assez souvent martelé mais il est absolument primordial que les données de votre application, qu'elles soient ou non sensibles, soient protégées convenablement pour éviter des attaques malveillantes ou pour se retrouver avec des fuites de données (ou des sauvegardes surprises décentralisées !). Vous devrez alors impérativement mettre en œuvre toutes les mesures nécessaires pour garantir la protection d'éléments déjà évoqués jusqu'ici : l'authentification, les autorisations, ou la protection contre les attaques faciles à exploiter dépendant du champ applicable à votre application.

3.2 Comment Nuxt peut-il être un bon candidat pour résoudre, répondre ou améliorer ces situations ?

En ce qui concerne du choix de la technologie, Nuxt est comme nous l'avons vu, un framework JavaScript conçu et écrit avec Vue comme support. Nuxt va venir surcharger le tout en offrant au framework diverses fonctionnalités pour le développement de votre application, qui pourront vous faire gagner du temps : la génération automatique de vos routes, le support natif du Server Side Rendering, qui améliorera significativement le SEO ou encore les différentes façons que vous avez de récupérer des ressources au-travers de différentes APIs.

Un des buts de Nuxt va consister à faire du développement et de la construction de votre application qu'il puisse être le plus facile et agréable possible, en proposant de hautes performances sans avoir besoin de jouer avec des paramètres et des configurations diverses et variées, comme on le ferait avec un serveur web tel qu'Apache ou Nginx, qui nécessite moultes configurations, notamment le serveur lui-même, la définition des différents site web, le réglage et l'activation / désactivation de plugins, etc.

3.2.1 L'expérience de développement et la préparation de l'application pour la mise en production

Nuxt bénéficie d'outils variés et performants qui peuvent sensiblement améliorer les performances de votre application.

Par défaut, Nuxt embarque ESBuild, Vite, PostCSS, Rollup et Webpack. Là où Vite va permettre le rechargement à chaud rapide de composant pendant le développement, permettant alors de gagner du temps à ne pas recharger la page et donc à devoir recharger l'ensemble des composants s'y trouvant, les autres outils vous permettront

d'avoir des bundles générés bien plus légers pour l'hébergement de votre application en production.

Webpack va jouer un rôle important dans ce processus puisqu'il permet notamment de regrouper différents types de fichiers (par exemple JavaScript, CSS ou des images) en un seul fichier optimisé. Il va également servir à la gestion des dépendances de votre application ou au principe de lazy loading, dont je parlerai plus tard.

Rollup va pour sa part agir comme rôle de réducteur de taille de fichier pour les dépendances que vous utiliserez. Il s'occupera de supprimer tous les morceaux de code non utilisés, permettant en principe de gagner en performances.

PostCSS va quant à lui faire des traitements d'analyse syntaxique du CSS de votre application. Il n'introduit donc pas de nouveaux éléments de syntaxes à apprendre comme le fait par exemple Sass.

Enfin, ESBuild s'occupe également de traiter des fichiers JavaScript. Il est connu pour les traites de façon très performante.

Tout comme WebPack, il est capable de transpiler, optimiser ou générer des bundles légers. WebPack et ESBuild sont utilisés conjointement pour leur polyvalence et leurs performances.

3.2.2 L'expérience utilisateur

Nuxt n'offre pas à proprement parler d'outils pour rendre vos interfaces plus navigables ou plus belles. À priori, aucun autre framework du même type que Nuxt non plus. Je ne parle donc pas ici des « frameworks » CSS. En revanche, comme je le disais, Nuxt a été conçu avec les dernières versions de Vue, lui offrant alors tout le meilleur que Vue peut donner : les composants réactifs, l'API composition, les performances optimisées, les directives Vue mais surtout tout l'écosystème lié.

Ainsi, l'expérience utilisateur au travers de Nuxt pourrait être améliorée par la façon dont vous écrirez votre code, en le rendant performant ou lent, mais moins vis-à-vis de son design.

3.2.3 Les performances de son moteur de serveur Nitro

En termes de performances, Nuxt est assorti d'un nouveau moteur de serveur : Nitro. Nous en parlerons plus tard mais ce nouveau moteur offre plusieurs axes d'améliorations par rapport au précédent au regard des performances qu'il délivre. Il joue un rôle essentiel concernant le Server Side Rendering, dont je parle également plus tard.

3.2.4 La maintenabilité et scalabilité

Concernant la maintenabilité et la scalabilité, la fonction d'auto-importation de composants a été étendue de sorte à inclure un nouveau dossier nommé « composables »[16]. Cette fonctionnalité vous permettra d'importer de façon automatique les différentes fonctions de l'API composition que vous pourriez définir dans ce dossier. Cette API est une fonctionnalité introduite dans Vue et permet la réutilisation de la logique de vos composants de façon plus flexible. En combinaison avec l'auto-importation, Nuxt pourra encore davantage faciliter la réutilisation de la logique de vos composants, ce qui contribue ici grandement au principe de scalabilité.

Le fait que vous puissiez auto-importer ces modules, composants et fonctions signifie que vous n'aurez plus besoin de les importer manuellement dans chaque fichier où vous les utiliseriez. Cela a pour conséquence directe de rendre votre code plus lisible et donc plus facile à maintenir, car vous n'aurez plus à vous soucier de gérer les importations dans votre application puisque Nuxt s'en occupera à votre place.

En plus de cela, Nuxt a été entièrement réécrit en TypeScript, ce qui lui confère tous les avantages dont j'ai également pu parler au chapitre 2.3. Cela ne pourra qu'améliorer votre expérience de développement, en particulier si l'IDE que vous utilisez, par exemple ceux de Jetbrains, est capable d'exploiter ce genre d'informations, pour fournir des fonctionnalités comme la détection d'erreur en continue ou l'auto-complétion, même si vous ne codez pas vos composants ou pages en TypeScript. Vous pourrez en effet bénéficier tout du long de l'ensemble de ces améliorer grâce au simple fait de l'infrastructure de Nuxt qui a été réécrite en TypeScript.

3.2.5 L'aspect souvent négligé : la sécurité

Nuxt offre au travers de son module nuxt-security différentes fonctionnalités qui viendront sécuriser votre application. Ce module a été notamment conçu pour s'occuper des dix menaces les plus importantes liées à Node.js et il ajoute des fonctionnements semblables à Helmet.js[17]. Ils servent respectivement comme plateforme logicielle libre permettant entre autres à faire fonctionner des frameworks et à sécuriser les requêtes http en définissant des entêtes de réponse http.

Ce module applique également deux nouveaux middlewares, définissant des entêtes de sécurité : un pour limiter le taux de requêtes (rate limiting) et un pour la taille desdites requêtes.

[16] https://nuxt.com/docs/guide/concepts/auto-imports
[17] https://nuxt.com/modules/security

« Intergiciel (Middleware) est un terme (défini vaguement) pour tout logiciel ou service permettant aux parties d'un système de communiquer et de gérer des données. C'est le logiciel qui gère la communication entre les composants et les entrées / sorties, de sorte que les développeurs peuvent se concentrer sur l'objectif spécifique de leur application.

Dans les frameworks d'application web côté serveur, le terme est souvent plus spécifiquement utilisé pour désigner des composants logiciels préconstruits pouvant être ajoutés au pipeline de traitement des requêtes / réponses du framework, pour gérer des tâches telles que l'accès à la base de données. »[18] - Wikipédia

Cette définition, très vague dans le cadre du développement d'application web, définit un ce qu'est un middleware. Dans le contexte de notre application Nuxt, un middleware va être un composant définit dans le dossier « middleware », qui va permettre d'intercepter une requête entre le moment où elle est exécutée et « vers quoi » elle veut aller. On a alors la possibilité de spécifier des comportements logiques permettant d'interdite à un utilisateur d'exécuter une requête ou de l'y autoriser.

Maintenant que cette notion est expliquée, revenons à nuxt-security. Un autre des comportements qu'il va faire va être d'appliquer plusieurs entêtes de sécurité par défaut, incluant notamment la CSP (Content-Security-Policy), le HSTS (Strict-Transport-Security) et le X-Frame-Options. Ils servent respectivement à définir quelle est la politique de votre application en matière d'accès aux contenus, à dire que votre application (via le navigateur) doit être accédé via une connexion HTTPS et enfin si votre application peut ou non être intégrée dans un « cadre de frame » (ou iframe) sur un autre site. Ces trois aspects contribuent à réduire la surface d'attaque et à protéger votre application.

Tous les aspects apportés par ce module peuvent être étendu et personnalisés selon le besoin que vous ou votre projet aura.

[18] https://developer.mozilla.org/fr/docs/Glossary/Middleware

4. Présentation approfondie de Nuxt 3

C'est dans ce chapitre que j'entre dans le sujet principal de ce travail de Bachelor. Je vais vous présenter quelques-unes des meilleures fonctionnalités apportées dans Nuxt et faire une petite analyse des différences avec la version précédente du framework.

4.1 Les fonctionnalités apportées par dans Nuxt

La dernière version majeure de Nuxt apporte son lot de fonctionnalités très intéressantes pour le développement d'applications web et touchent à l'ensemble des principes et sujets que j'ai déjà pu aborder dans le cadre de ce travail de Bachelor et auxquels vous pourrez être confronté : les performances, la facilité de développement, le SEO, la modernité du style de développement, la maintenabilité du code dans le temps, etc.

Toutes les fonctionnalités présentées ne sont pas toujours du « fait maison » mais sont un agrégat de bibliothèques qui ont été réunies de sorte à fournir un framework aussi complet que possible.

4.1.1 Le Hot module reloading (ou rechargement de module à chaud)

La première fonctionnalité intéressante que Nuxt offre est le rechargement à chaud de modules[19]. Ce principe peut être expliqué en prenant le cas suivant : vous amenez votre voiture pour une réparation chez votre mécanicien et, au lieu de devoir arrêter votre moteur, retirer la clef, sortir de votre voiture, la verrouiller puis enfin attendre qu'elle soit réparée, le mécanicien vous dit qu'il peut effectuer en temps réel les opérations nécessaires à la réparation, sans impacter le reste de votre véhicule, vous permettant alors de continuer à rouler.

Dans le contexte du développement de votre application Nuxt, le hot module reloading vous permet d'effectuer toutes les modifications que vous souhaitez dans votre code sans devoir recharger complètement votre page en la rafraîchissant pour y voir les changements. Les modules sur lesquels vous aurez travaillé auront localement été rechargés en direct, à chaud. De cette façon, vous pouvez voir en tout temps les modifications que vous apportez à votre code, sans avoir besoin d'appuyer frénétiquement sur votre touche rafraîchissant votre page web. Cela vous permettra de gagner un temps non négligeable.

Supposez par exemple que vous souhaitiez modifier une partie du CSS de votre application, ou encore que vous changiez la hiérarchie des éléments d'une page. Vous verrez instantanément les changements sur votre navigateur. Vous pouvez ainsi vous

[19] https://nuxt.com/docs/guide/concepts/server-engine#server-engine

concentrez entièrement au développement et à l'efficacité de votre code, plutôt que de devoir perdre votre temps à devoir rafraichir votre page.

4.1.2 Le server-side rendering (ou SSR)

Un autre atout important qu'offre Nuxt est la haute performance applicative qu'il offre dans un environnement de production avec la génération par défaut du rendu côté serveur (SSR). [20]

L'avantage principal que l'on peut noter sur le SSR est qu'il améliore grandement les performances et l'expérience d'un utilisateur qui passerait sur votre application. En effet, avec le SSR, le serveur qui héberge l'application va générer une version HTML statique de votre application qui sera envoyée à votre utilisateur lorsqu'il en fait la demande. Cela permet d'éviter d'attendre que tout le code JavaScript de votre page soit téléchargé sur le navigateur de votre utilisateur et qu'il doive attendre de l'avoir exécuté entièrement avant d'afficher quoi que ce soit sur son écran. Le temps de chargement initial est alors nettement réduit lors de l'affichage de la première page. Cela améliore l'expérience offerte à votre utilisateur en lui donnant un sentiment de rapidité.

L'autre atout qu'offre le SSR est le fait qu'il peut vous aider à améliorer le référencement naturel (SEO) de vos applications par les moteurs de recherche car il permet une indexation plus facile de vos pages, du fait que les bots n'ont pas besoin d'exécuter le code JavaScript de la page pour afficher ladite page, puisqu'elle a déjà été préchargée du côté serveur.

En raison de de son intégration native, il ne vous sera demandé aucun effort supplémentaire pour une mise en place du SSR. Nuxt s'occupera pour vous, de façon transparente, de générer les pages et composants que vous écrirez.

En outre, combiné aux fonctionnalités dynamiques du client side rendering, Nuxt pourra ajouter de l'interactivité et de la dynamique aux pages générées avec le SSR. Cela offrira une expérience utilisateur fluide, en maintenant les avantages du SSR en termes de performances et de référencement.

De plus, la gestion des données traitées de manière asynchrone est améliorée avec le SSR de Nuxt. Différentes fonctions dont nous reparlerons plus tard, telles qu'UseAsyncData ou UseFetch vous permettront de récupérer et précharger des données avant même de générer les pages faites par le SSR. Cela a pour avantage de garantir que l'ensemble des données soient disponibles dès le rendu initial de vos pages.

[20] https://nuxt.com/docs/guide/concepts/rendering

Cela offre une expérience utilisateur toujours plus riche car elle évite d'éventuels temps de chargement supplémentaires.

Pour finir de présenter le SSR, laissez-moi vous présenter les différentes formes de SSR que Nuxt propose : le Universal Rendering (ou rendu universel) est celui activé par défaut. Vous avez également le Client-Side Rendering (ou rendu côté client), l'Hybrid Rendering (ou rendu hybride) et enfin le Edge-Side Rendering (ou rendu côté serveur)[21]. Le choix du type de rendu dépendra de plusieurs critères de sélection :

- Votre besoin en interactivité : si la page que vous construisez nécessite par exemple une grande interactivité du côté de l'utilisateur – du client – avec des transitions de page, des mises à jour en temps réel, etc. alors le rendu côté client est évidemment plus adapté. Cela a un effet bénéfique vis-à-vis de Vue.js, car vous lui donnez alors le contrôle de votre page et lui permettez de gérer les interactions en question
- La question de la performance : si le but recherché est une expérience utilisateur rapide et performante, alors c'est le rendu universel qui devra être privilégié. Cette architecture implique le serveur et le client et permet notamment d'afficher rapidement le contenu statique de votre page au premier chargement. Tout le côté « interactif » est ensuite géré côté client.
- SEO et indexation de pages par les moteurs de recherche : si l'un des objectifs de votre site ou application soit qu'il puisse être correctement référencé par les moteurs de recherche, alors le choix du rendu universel sera là aussi privilégié en raison du fait que le contenu lu sera celui qui aura été généré côté serveur.
- La complexité du développement de votre site / app : le serveur et le client sont bien évidemment deux environnements bien différents. Le rendu universel n'est alors peut-être pas le meilleur des choix – toutes proportions gardées – en termes de complexité de développement. Vous devrez en effet prendre en compte ces deux environnements différents lors du développement de votre app. Si on prend le rendu côté client, il peut parfois être plus simple à appréhender, du fait que vous savez d'avance que tout le code que vous écrirez ne s'exécutera que du côté client.
- Les coûts : vous l'aurez probablement deviné, un système de rendu impliquant un serveur entraîne forcément des coûts liés à un hébergement acceptant d'accueillir une application tournant avec Nuxt (node par exemple). La question des performances du serveur peut aussi alourdir les frais d'hébergement. Le rendu côté client, lui, a moins cet inconvénient, si on peut le qualifier comme tel, car il peut être hébergé sur n'importe quel type de serveur statique.
- Enfin, le critère de la fréquence de visite de vos utilisateurs : si votre site est visité souvent et que les pages de votre app ne nécessitent pas d'indexation par les moteurs de recherche, le rendu côté client est peut-être plus adapté. Vous pourrez en outre mettre en cache lesdites pages enfin de rendre la navigation encore plus rapide.

[21] https://nuxt.com/docs/guide/concepts/rendering

4.1.3 Le déploiement « on the edge » (déploitement CDN)

Nuxt vous propose également la possibilité de faire du déploiement « on the edge ». [22]Il s'agit d'une approche de déploiement d'applications aux « marges » d'un réseau de distribution de contenu, plus communément connus sous l'appellation de CDN. Cela remplace le déploiement dit « au centre » du réseau. On utilise cette technique pour améliorer les performances d'une application en permettant de placer l'application au plus proche de l'endroit où la demande est générée : proche de l'utilisateur. Dans le contexte d'une application Nuxt, cela vous permet – si tant est que vous souhaitiez la déployer sur un CDN – de réduire considérablement les temps de chargement et de réduire la possibilité de potentielles et redoutées attaques par déni de service. En effet, l'ensemble des serveurs abritant votre application sont distribués et répartis sur une vaste étendue du réseau.

Cela permet également de simplifier la gestion de la mise en cache de fichiers statiques, permettant là aussi de réduire les temps de chargement, donc d'améliorer la satisfaction de l'utilisateur en ne le frustrant pas par une attente avec le chargement de votre application. Nuxt permet le déploiement on the edge avec des configurations extrêmement basiques chez des fournisseurs tels qu'AWS, Azure, CloudFlare ou encore Firebase.

4.1.4 L'automatisations du framework et ses conventions

Nuxt se base sur des conventions et une structure de dossiers pour ses fichiers de sorte que vous puissiez vous concentrez uniquement sur la création et l'écriture de fonctionnalités. Vous vous en avez le besoin express, vous aurez pour cet aspect la possibilité d'écraser toute cette logique de base en modifiant ces comportements par défaut dans le fichier de configuration de votre application.

4.1.4.1 Génération automatique de vos routes

À la différence d'autres frameworks JavaScript, Nuxt propose une génération des URLs (les routes) de votre application basée sur la structure des fichiers et dossiers du répertoire « pages » de l'application[23]. Cela vous permet d'organiser assez naturellement votre application et de gagner du temps en n'ayant pas besoin d'aller écrire à la main votre fichier de routes.

Ce système fonctionne sur la base du nommage des dossiers et fichiers .vue contenus dans le dossier « pages » de votre application.

[22] https://nuxt.com/docs/getting-started/deployment
[23] https://nuxt.com/docs/getting-started/routing

Prenons la structure suivante pour exemple :

Figure 6 : exemple simple de structure de pages d'une application Nuxt

(Evan Schleret)

Basé sur cet exemple, Nuxt va automatiquement créer les routes <domaine de votre app>/ et <domaine de votre app>/account, sans avoir besoin de les déclarer. Il n'est pas nécessaire d'explicitement écrire le chemin /index dans votre barre d'URL car car le fichier index.vue est traité comme un « / ».

Les possibilités liées au routage de Nuxt ne s'arrêtent évidemment pas là.

En effet, la structure présentée précédemment ne tient pas compte des cas où vous voudriez définir des paramètres dans vos URLs, pour afficher par exemple les informations spécifiques d'un utilisateur (une route « /utilisateurs/<un utilisateurs> »). Nuxt vient répondre à ce besoin en vous proposant d'ajouter dans le nom de vos dossiers et fichiers .vue un nom entre crochets carrés (« [ ] »). Cela vous permettra de rendre vos routes dynamiques.

Figure 7 : exemple de structure de page avec un paramètre dynamique dans une application Nuxt

(Evan Schleret)

Dans cet exemple, le paramètre « [date] » est passé dans toutes les pages contenues dans ce dossier. Vous pourrez ensuite récupérer ce même paramètre dans votre page avec la fonction useRoute() de Nuxt, qui permet d'interagir avec les URLs.

Un autre aspect lié aux routes est la possibilité de définir une route qui « attrape tout ». Une sorte de route « joker » qui permet de répondre à tout ce qui pourrait être passé dans l'URL. Pour ce faire, vous n'avez qu'à rajouter trois petits points avant le nom de votre paramètre : […date]. À partir de ça, votre URL, en reprenant l'exemple précédent, l'URL pourrait alors très bien être /blog/quelque/chose ou encore /blog/encore/autre/chose.

4.1.4.2 Auto-importation des composants

Un autre atout de Nuxt concerne tout son système d'auto-importation de vos composants[24]. Nuxt va en effet vous permettre de ne jamais avoir besoin d'importer explicitement une fonction, méthode ou composant dans vos différentes pages.

S'ils sont placés correctement dans les bons répertoires, Nuxt vous permettra alors de ne jamais vous soucier de cet aspect du développement de votre application.

Tout ce système contribue lui aussi à réduire la complexité et la durée nécessaire au développement. En n'ayant pas besoin de vous souciez de comment et quand importer vos fonctions et modules, vous pouvez maintenant vous concentrer sur le code.

L'auto-importation faite par Nuxt permet également une meilleure organisation et lisibilité du code. En outre, les performances sont améliorées par le fait que les bundles générés

[24] https://nuxt.com/docs/guide/concepts/auto-imports

pour la production sont plus légers, du fait que seules les parties utilisées dans votre code sont incluses dans votre application.

Enfin, vous apprécierez cette fonctionnalité par le simple fait qu'elle vous retire l'exécution d'une tâche répétitive et peu intéressante.

4.1.5 Les façons de récupérer des données sur des API

Nuxt propose plusieurs méthodes pour récupérer des données via des APIs notamment. La méthode utilisée est dépendante du scénario et des besoins de la page. Vous retrouverez les méthodes useFetch, useLazyFetch, useAsyncData et useLazyAsyncData[25]. Ces méthodes permettent toutes de récupérer des données sur n'importe quelle URL, qu'elle soit propre à votre application ou pas. Nuxt permet par ailleurs d'utiliser ces méthodes dans d'autres endroits que vos pages ; vous pouvez également les utiliser dans vos différents composants et plugins.

Penchons-nous plus en détail sur les spécificités techniques des deux premières méthodes. useFetch et useLazyFetch sont utilisées pour récupérer des données de façon synchrone, c'est-à-dire que votre code est exécuté de manière séquentielle, à la façon d'un script qui s'exécuterait ligne par ligne. Si une partie de ce code prend plus de temps à s'exécuter – par exemple dans le cas d'une récupération de données sur une API – alors l'exécution de votre code « s'arrêtera » tant que la ligne en cours d'exécution n'aura pas terminer de s'exécuter. Dans le cas d'une application Nuxt, la page sur laquelle l'utilisateur est n'affichera aucune progression, ce qui pourra laisser penser à un blocage, ce qui pourrait nuire à l'expérience que cet utilisateur aura sur votre application.

useAsyncData et useLazyAsyncData, elles, permettent de récupérer ces données de façon asynchrone : l'exécution du code de votre composant, de votre page ou de votre plugin ne sera pas bloquée par un temps d'exécution plus long lors de la récupération des données.

Nous verrons qu'il y a en réalité une petite subtilité permettant tout de même d'utiliser useFetch sans bloquer l'exécution de votre code… Avant cela, décrivons davantage chacune de ces méthodes.

useFetch, base des méthodes pour récupérer des données, est là pour uniformiser la façon dont cette récupération est faite. Lorsque vous souhaiterez utiliser cette méthode, vous devrez fournir l'URL souhaitée ainsi que les paramètres de récupération voulus, comme la méthode http utilisée, les entêtes, les paramètres de l'URL, etc. Une fois

[25] https://nuxt.com/docs/getting-started/data-fetching

exécutée, la méthode génèrera une clef unique basée sur ces paramètres. Cette clef sera utilisée pour stocker en mémoire cache l'ensemble des données récupérées, de sorte à éviter des appels à l'API qui ne seraient pas utiles car les données générées seraient à priori les mêmes.

useLazyFetch est très similaire à useFetch, à la différence qu'une option « lazy » est activée. Cette option est faite de sorte que la méthode de récupération des données ne bloque pas l'exécution de la page, du composant ou du plugin. Cette méthode est exécutée de manière asynchrone. Il est donc important de noter qu'il faudra gérer le cas où les données rendues par l'API seraient nulles.

useAsyncData, comme brièvement décrit, est utilisée pour récupérer des données de façon asynchrone. Elle est donc très similaire à useFetch, à la différence qu'elle répond à des besoins plus complexes et moins courant que useFetch. Par moins courant, on peut par exemple penser au cas où on voudrait un contrôle bien plus important sur la façon et sur le moment où les données sont récupérer ou encore comment elles sont traitées avant d'être renvoyées à l'utilisateur. Dans un autre cas, on peut imaginer qu'il vous soit nécessaire que plusieurs requêtes se termine avant d'afficher un retour à l'utilisateur.

Ces cas étant en général moins courant, useFetch a été créé en ajoutant une couche d'abstraction supplémentaire de sorte que vous n'ayez pas besoin de vous soucier de tout cela.

useLazyAsyncData est comme vous pouvez l'imaginer, similaire à useAsyncData, également avec l'option lazy activée. Comme pour useLazyFetch, vous devrez alors traiter le cas où les données retournées par votre API seraient nulles.

4.1.6 Nitro : le nouveau moteur de serveur de Nuxt

Nuxt s'est équipé d'un nouveau moteur de serveur, appelé Nitro. Nitro est développé par une équipe connue de la communauté, unjs (Unified JavaScript Tools)[26]. Ils sont notamment les créateurs de repos Github comme ofetch.

Nitro offre plusieurs fonctionnalités qui répondent de façon différente que beaucoup d'autres serveurs, écrits ou pensés sur des approches qu'on pourrait qualifier de plus traditionnelles. Nitro a par exemple été conçu pour tourner sur plusieurs types de plateformes différentes, notamment Node.js, deno ou d'autres plateformes serverless comme CloudFlare Workers.

[26] https://nitro.unjs.io/

Nitro permet également la création et l'intégration beaucoup plus facile d'API car il prend en charge nativement les routes d'API que vous aurez l'occasion de créer, vous offrant la possibilité de créer des logiques directement exposables à votre application Nuxt, sans devoir passer par une solution annexe comme un autre framework. Cette utilisation dépend évidemment de vos besoins et possibilités liés aux projets.

Nitro est également pourvu d'une fonctionnalité de « code splitting », permettant de découper du code en différents sous-fichiers, ce qui a pour effet direct de garantie en matière d'augmentation de performances pour votre application.

J'en parlais dans le passage lié au SSR et au hot module replacement, mais je ne mentionnais pas encore Nitro est la partie de Nuxt qui fournit ces fonctionnalités.

4.2 Analyse des différences avec la version majeure précédente (Nuxt 2)

Nuxt fournit sur son site un tableau comparatif nous permettant de décrire assez rapidement les points importants qui ont été changés entre les versions 2 et 3 du framework.[27]

[27] https://nuxt.com/docs/getting-started/upgrade

Figure 8 : tableau comparatif des différences notables entre Nuxt 2 et 3

Feature / Version	Nuxt 2	Nuxt Bridge	Nuxt 3
Vue	2	2	3
Stability	😊 Stable	😌 Semi-stable	😊 Stable
Performance	🏎️ Fast	✈️ Faster	🚀 Fastest
Nitro Engine	❌	✅	✅
ESM support	🌙 Partial	👍 Better	✅
TypeScript	☑️ Opt-in	🚧 Partial	✅
Composition API	❌	🚧 Partial	✅
Options API	✅	✅	✅
Components Auto Import	✅	✅	✅
`<script setup>` syntax	❌	🚧 Partial	✅
Auto Imports	❌	✅	✅
webpack	4	4	5
Vite	⚠️ Partial	🚧 Partial	✅
Nuxi CLI	❌ Old	✅ nuxi	✅ nuxi
Static sites	✅	✅	✅

(Site officiel du framework Nuxt)

Il résume plusieurs points déjà abordés jusqu'ici : le support de Vue 3, tout l'aspect lié aux performances qu'offrira votre application, le nouveau serveur Nitro Engine, le fait que TypeScript soit supporté (et accessoirement que le framework ait été entièrement réécrit en ce langage), l'API composition, la syntaxe script setup, l'auto-importation généralisée, la version de webpack 5 et vite.

Mention honorable à l'outil de cli (interface de ligne de commande) qui présente des aspects pratiques comme la génération de plugin, layout, composants, etc.

5. Conception de l'application

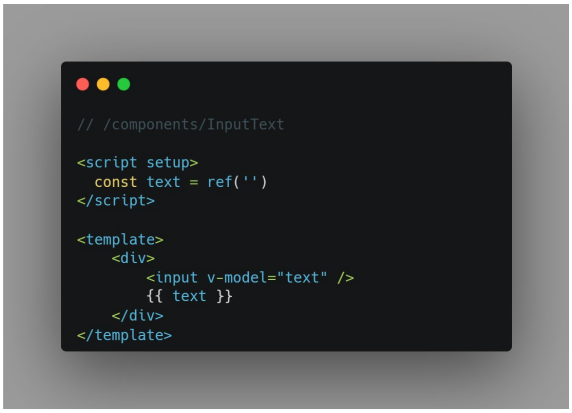
5.1 Description de l'architecture de mon application

Comme j'ai pu l'expliquer précédemment, Nuxt utilise une architecture dite en « composants ». Cela signifie que les composants que vous écrirez pourront être réutilisés. C'est même la logique qui réside dans Nuxt. Plutôt que de faire de longues pages avec des parties de code qui peuvent être extraites.

De plus, un composant n'est pas un simple morceau de code qu'on peut réutiliser. En effet, il faut considérer que chaque représentation d'un composant sur une page est indépendante des autres mêmes représentations de ces composants présents également sur cette même page.

Laissez-moi vous expliquer ce concept en prenant l'exemple suivant : imaginez que votre page contienne un composant qui s'appellerait « InputText », comme celui-ci-dessous.

Figure 9 : exemple de composant Nuxt



(Evan Schleret)

Et que ce composant InputText soit présent trois fois sur la page.

Figure 10 : exemple de page intégrant plusieurs fois un même composant Nuxt

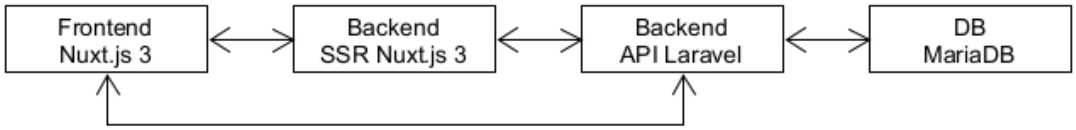
(Evan Schleret)

Le composant se contente ici d'afficher ce que l'utilisateur tape dans le champ de formulaire et ce composant est affiché trois fois sur la page. L'utilisateur pourra donc taper trois textes différents, sans que le texte ne se modifie dans les trois champs en même temps, puisque chaque composant est indépendant. Le composant InputText est affiché trois fois. Pourtant, chacun d'entre eux est indépendant. Cet aspect présente deux avantages en même temps : la réutilisabilité et la modularité.

En ce qui concerne de l'architecture plus globale de mon application, elle se sépare en deux couches : le frontend avec Nuxt et le backend avec une API REST Laravel, qui gère toute la partie backend de l'authentification et la gestion des permissions, de la sécurité et la gestion des données. Le backend s'occupe donc également de la partie dite de persistance des données (base de données). Dans le cadre de ce travail de bachelor, je n'entrerai pas en détail concernant cette partie.

Le mode de SSR de mon application Nuxt est configuré en mode universel. Cela implique que même si Nuxt agit comme frontend, il convient de représenter une autre couche de backend en raison du travail de génération de pages dont mon serveur hébergeant l'application s'occupe.

Figure 11 : représentation de l'architecture de mon application Nuxt



(Evan Schleret)

Le frontend va donc afficher les pages générées par le backend de Nuxt, qui s'occupe lui-même d'échanger avec l'API et qui génère les pages. Notez que le frontend peut aussi faire des appels API au travers de requêtes XHR (AJAX).

5.2 Justification des choix technologiques

J'ai décidé de justifier le choix de Nuxt comme la technologie choisie pour mon frontend. Cinq raisons d'une liste non exhaustive justifient pour moi ce choix : l'écosystème de Nuxt au travers de Vue.js, le SSR, le système d'auto-importation dont nous avons déjà un peu parlé, le routage automatique et l'intégration facile de bibliothèques externes. Bien sûr, l'idée n'est pas de répéter les explications des concepts en question que j'ai déjà eu l'occasion d'expliquer assez largement, mais bien de souligner les idées clefs.

5.2.1 L'ecosystème de Nuxt au travers de celui de Vue

Comme répété plusieurs fois jusqu'ici, Nuxt 3 est basé sur Vue 3. Il bénéficie alors de tout son écosystème. En prenant Nuxt comme framework, vous pourrez alors exploiter pleinement tout ce que Vue a à offrir, en plus de bénéficier des nombreux avantages instaurés par Nuxt : son système de routage, son système de « code splitting », l'auto-importation de modules et composants, etc.

Au-delà de cet argument, Nuxt bénéficie de son propre écosystème, grâce aux nombreux modules installables[28] très simplement, mais nous en parlerons au dernier point.

5.2.2 Le SSR

L'aspect du SSR de Nuxt est très important dans le choix de ce framework pour vos futurs projets web. Du SSR de Nuxt, on peut retenir plusieurs aspects principaux :

- Le rendu universel, activé par défaut sur toutes nouvelles installations, présente des avantages majeurs : une meilleure expérience pour vos utilisateurs, des temps de chargement nettement diminués et une optimisation beaucoup plus grande en matière d'indexation de contenus par les moteurs de recherches.
- Toujours grâce au rendu universel, les performances sont accrues. En effet, le fait que le contenu statique soit préalablement généré par le serveur permet forcément d'afficher plus rapidement ledit contenu ; le navigateur affichant plus vite du contenu statique que du contenu généré par JavaScript. Cela permet, encore, de contribuer à une expérience utilisateur plus fluide et agréable. Cela dépendant en revanche évidemment des performances du serveur qui héberge l'application. Si celui-ci tourne sur un CPU de très vieille génération, il ne faudra pas s'étonner si les performances ne sont pas au rendez-vous.
- Je n'aurai de cesse de le dire mais le SEO est grandement optimisé avec le rendu universel proposé par Nuxt. La version statique de votre page étant préalablement été générée par le serveur, celle-ci peut être lue en intégralité par le bot du moteur de recherche qui l'analysera et l'indexera.
- De part ces quatre différents modes de rendu différents, Nuxt peut répondre à tout type de scénarios et de cas d'utilisation. En effet, que vous souhaitiez développer votre blog, votre forum, votre boutique en ligne ou simplement votre site internet, Nuxt peut répondre à tous les cas d'utilisation du fait des quatre types de rendu différents.
- Nuxt introduit un principe s'appelant « Hydratation ». Ce principe, ou processus, va permettre de rendre toutes les pages statiques quand même interactives, une fois téléchargées sur le navigateur de votre utilisateur. Cela vous permet de garder l'aspect interactif et dynamique des applications JavaScript.
- Nuxt propose pour ceux qui en ont le besoin, la possibilité de faire un rendu côté serveurs sur des serveurs de type CDN Edge, répondant aux questions relatives à la latence et aux performances, en permettant respectivement de la réduire et de l'augmenter.

[28] https://nuxt.com/modules

C’est l’ensemble de ces points qui font du SSR de Nuxt un super allié pour la performance de vos applications. Vous aurez le choix de configurer quel mode de SSR vous souhaitez, sans devoir vous séparer de Nuxt en raison d’incompatibilité lié au SSR.

5.2.3 Le système d’auto-importation

De la même façon que je l’ai abordé au point précédent, nous pouvons retenir plusieurs aspects importants autour du système d’auto-importation de Nuxt :

- Premièrement, Nuxt apporte de la simplicité. En effet, en raison du fait que vous n’avez pas besoin d’importer explicitement vos composants, méthodes, et fonctions dans les pages de votre application, vous gagnez du temps en supprimant de votre processus de développement toutes les fois où vous avez à résoudre un problème d’importation.
- Secondement, grâce à la technique de tree shaking, votre code est allégé de tout le code inutile dont vous ne vous servez pas ou qui est redondant. Ça permet par la même occasion de rendre votre application plus performante car moins lourde donc plus rapide, en principe, à charger.
- Relié au premier point, les risques d’erreur liés à des importations incorrectes sont réduits. En effet, vous n’avez qu’à vous concentrer sur la partie strictement applicative et non plus sur le détail.
- À l’inverse des frameworks nécessitant une déclaration dite globale plus classique, Nuxt permet de conserver tous les « typings » et les complétions faites par les IDE modernes, comme ceux de la suite Jetbrains, pour les fonctions et auto-imports. Cela permet là aussi d’améliorer le développement car on peut ainsi conserver l’auto-complétion faite par les IDE.

En bref, vous gagnez au niveau du temps de développement, un risque diminué en matière d’auto-importation, vous gagnez en performance grâce au tree shaking et en maintenance de votre code.

5.2.4 Le routage dynamique

Cette partie est moins longue que d’autres mais le routage généré automatiquement par Nuxt fait de cette fonctionnalité un vrai gain de temps et d’organisation. En effet, le fait de devoir baser la hiérarchie de ses dossiers et vues dans le dossier pages de votre application fait que vous êtes obligé de vous questionner sur quelle pourra être la structure idéale et logique.

De plus, le composant NuxtLink[29] qui permet la navigation entre les différentes pages est muni de plusieurs paramètres répondant à tous les cas d’utilisations que vous pourriez avoir. Vous pourrez également définir des attributs HTML classiques tels que target ou rel.

La facilité d’écriture de middlewares de routes fait également du routage de Nuxt un fort atout.

[29] https://nuxt.com/docs/api/components/nuxt-link

5.2.5 L’intégration de bibliothèques

Je l’ai déjà dit précédemment mais Nuxt est un framework dont la popularité ne cesse d’augmenter, comme avait pu le faire le framework Laravel il y a quelques années, aujourd’hui projet ayant le plus d’étoiles sur Github dans la catégorie des frameworks PHP[30].

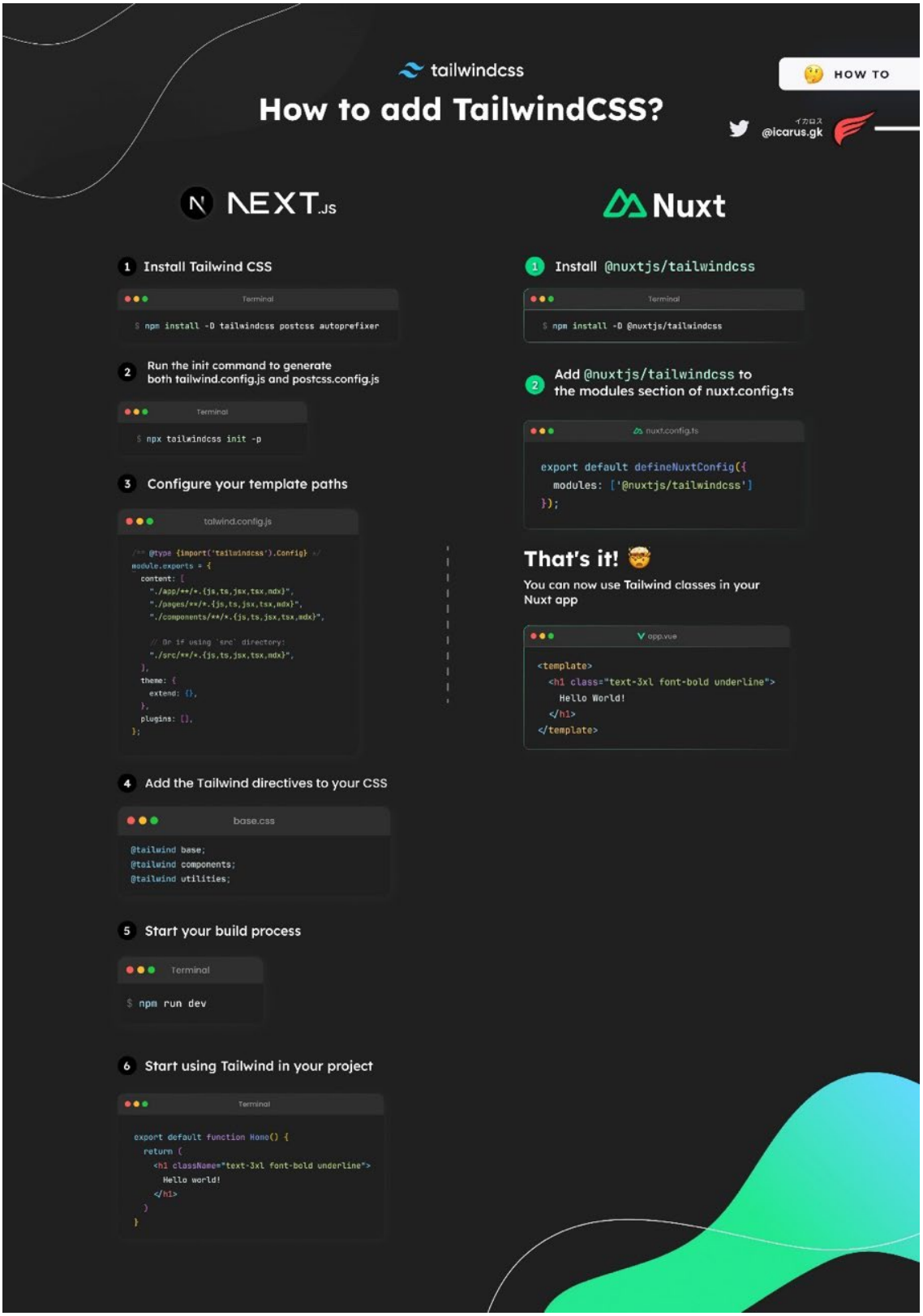
Plusieurs bibliothèques, appelées également modules sur le framework, ont été adaptées pour Nuxt et sont aujourd’hui installables de façon bien moins complexes et longues que dans d’autres frameworks, à l’image de tailwindcss, librairie très connue, utilisée comme alternative à d’autres bibliothèques CSS, offrant bien plus de possibilités et de libertés.

Vous pourrez voir par vous-même à la Figure 12 la simplicité par laquelle on peut installer cette bibliothèque dans Nuxt. C’est avec son framework paronyme, Next.js, que cette installation est comparée.

Au-delà de la bibliothèque tailwincss, de nombreux autres modules sont poussés et mis en avant sur le site de Nuxt. On y trouve différentes catégories représentant assez, comme j’ai pu le dire, les différents aspects auxquels nous pouvons être confrontés lors du développement d’une application web : des outils analytiques, une catégorie dédiée au CSS, une autre aux bases de données, aux outils pour les développeurs ou aux polices d’écritures. Pour des sites impliquant un aspect de commerce, il y a également deux catégories dédiées : ecommerce et paiement. Enfin, pour finir de ne citer que les « plus importantes », une catégorie liée aux performances, à la sécurité et au SEO.

[30] https://Github.com/EvanLi/Github-Ranking/blob/master/Top100/PHP.md

Figure 12 : schéma de comparaison sur l'installation de tailwindcss sur le framework Next.js et Nuxt



(Tweet publié par l'utilisateur @icarusgkx, 14 juin 2023)

Comme vous pouvez le constater, dans Nuxt, nous n'avons besoin que de deux étapes simples pour installer et utiliser tailwindcss dans une application Nuxt. Aucune configuration supplémentaire n'est requise en dehors de l'installation et la déclaration du module dans la config de votre application. Cet exemple n'en est qu'un représentant toute la simplicité de l'installation et de l'utilisation de bibliothèques avec Nuxt.

Beaucoup des modules proposés et mis en avant sur le site de Nuxt suivent cette logique de simplicité, ce qui ne manquera certainement pas de vous faire gagner du temps pour mieux le dépenser à réellement produire du code.

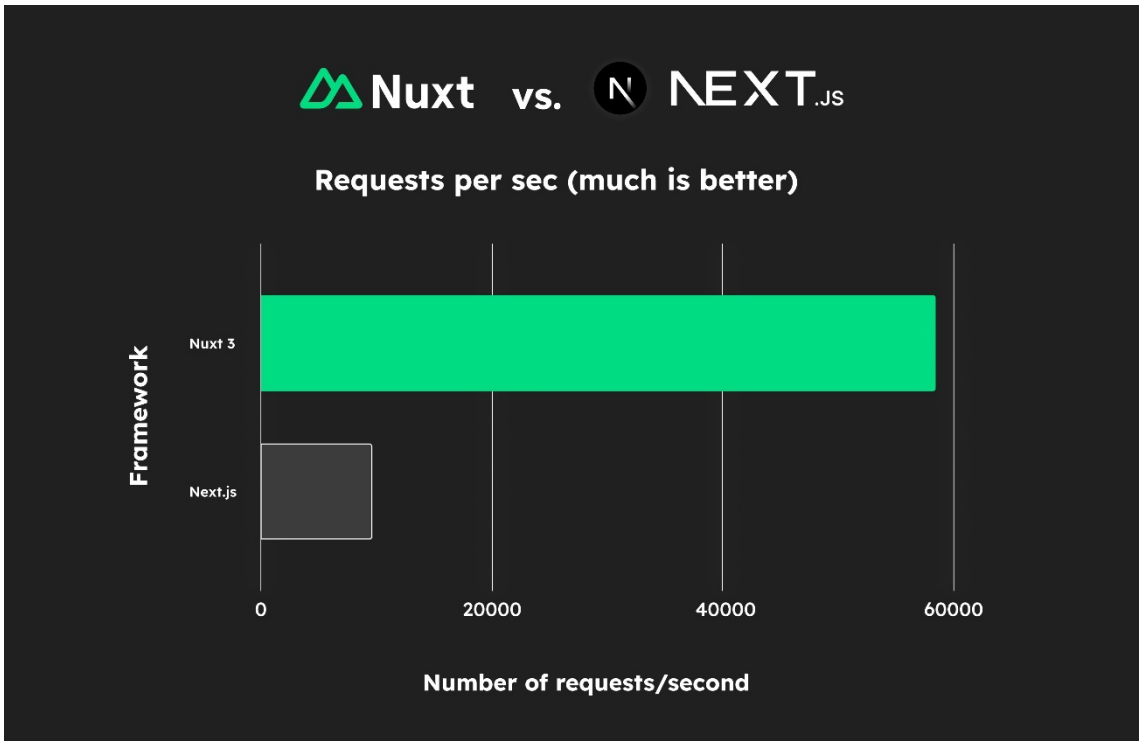
6. Évaluation des performances de Nuxt

J'ai trouvé deux développeurs sur Twitter que se sont déjà un peu penché sur la question, en comparant Nuxt avec le framework Next.js. Les deux applications faites par les développeurs ont été testée avec oha[31], un petit programme pour tester la charge qu'accepte une application.

Dans le benchmark de vedantnn71[32], fait sur dix secondes pour les deux metrics, Nuxt a pu absorber 61'354 requêtes par secondes et Next a pu en prendre 11'349. Nuxt a donc pu répondre à 540% plus de requêtes que son paronyme. Quant au temps moyen de réponse, il était de 0.0081 seconde pour Nuxt et de 0.044 seconde pour Next. Next met donc en moyenne 5.4 fois plus de temps que Nuxt pour répondre aux requêtes.

L'application a été testée sur la réponse à l'affichage d'un « Hello World ».

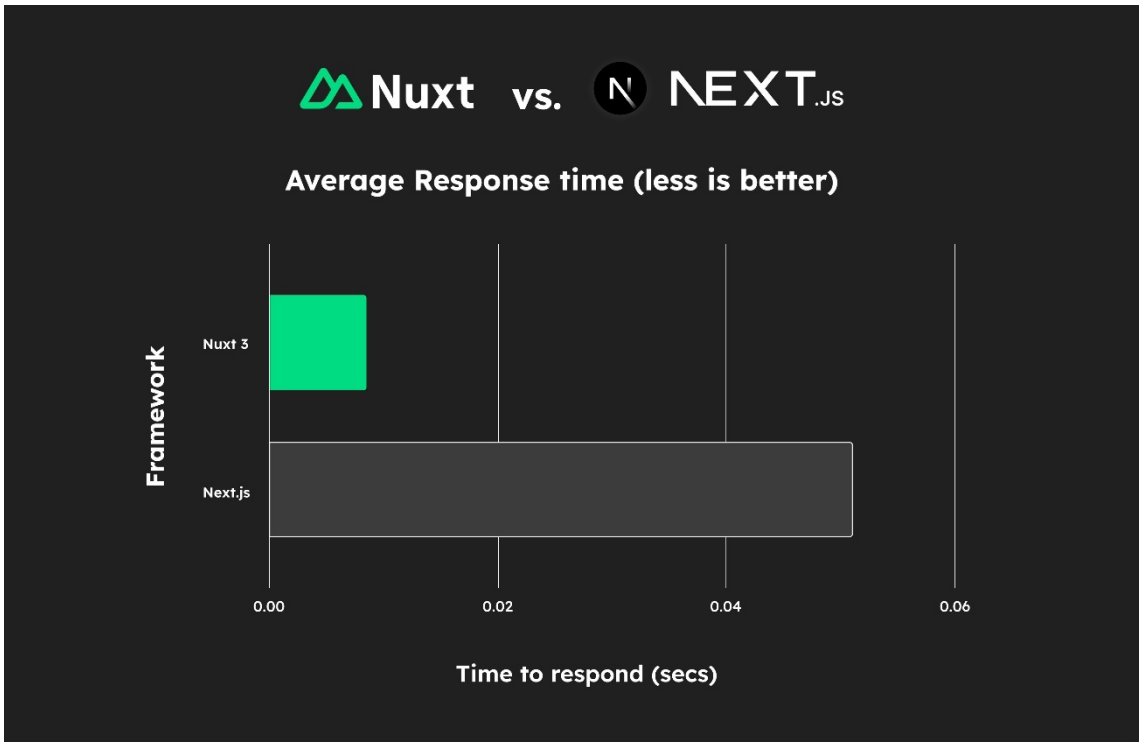
Figure 13 : comparatif entre Nuxt et Next en nombre de requêtes par seconde



(vedantnn71, repo Github « nuxt-next-perf-test »)

[31] https://github.com/hatoo/oha
[32] https://github.com/vedantnn71/nuxt-next-perf-test

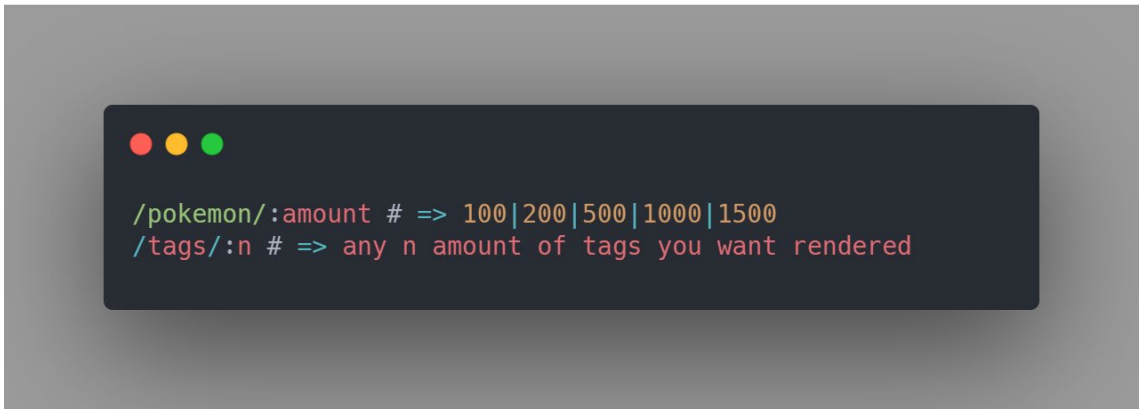
Figure 14 : temps moyen de réponse entre Nuxt et Next



(vedantnn71, repo Github « nuxt-next-perf-test »)

Le benchmark d'icarusgk[33] a également été testé sur un « Hello World » mais également sur l'accès via des routes dynamiques affichant n nombre de tags.

Figure 15 : routes créées pour le benchmark d'icarusgk

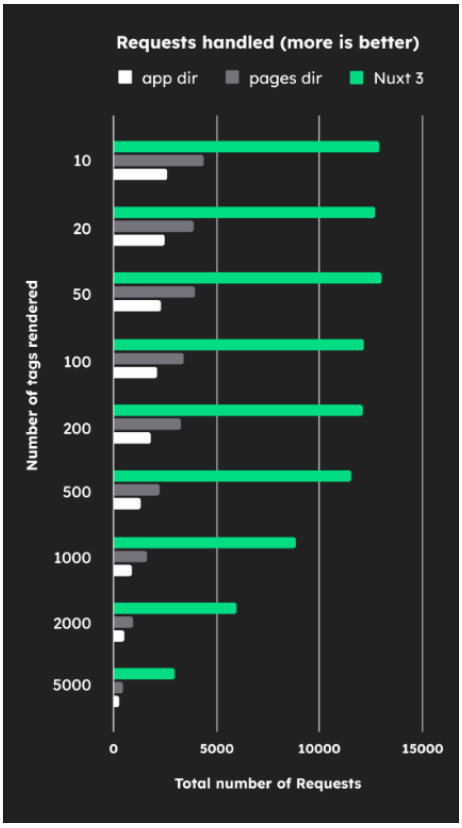


(icarusgk, repo Github « fw-bench »)

Les résultats sont là aussi impressionnants, Nuxt batant clairement Next, que ce soit en nombre de requêtes acceptées par secondes en fonction du nombre de tags affichés, de temps moyen de réponse ou de nombre de requêtes sur le « Hello World ».

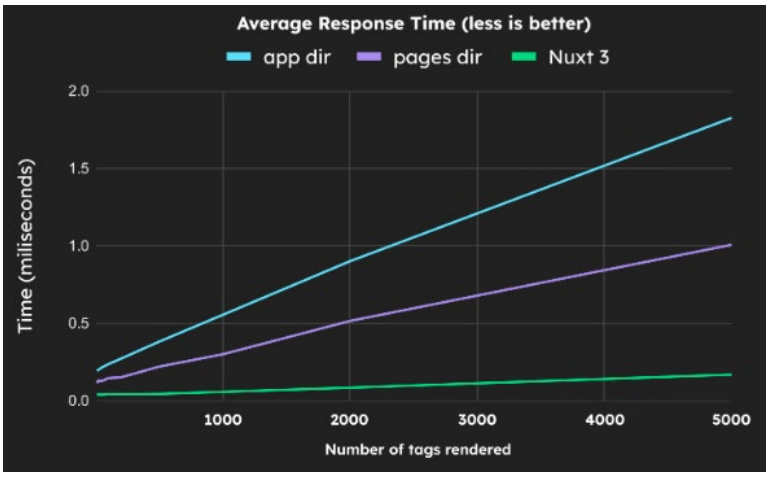
[33] https://github.com/icarusgk/fw-bench

Figure 16 : comparatif sur le nombre de requêtes par secondes absorbées par Nuxt et Next en fonction du nombre de tags affichés



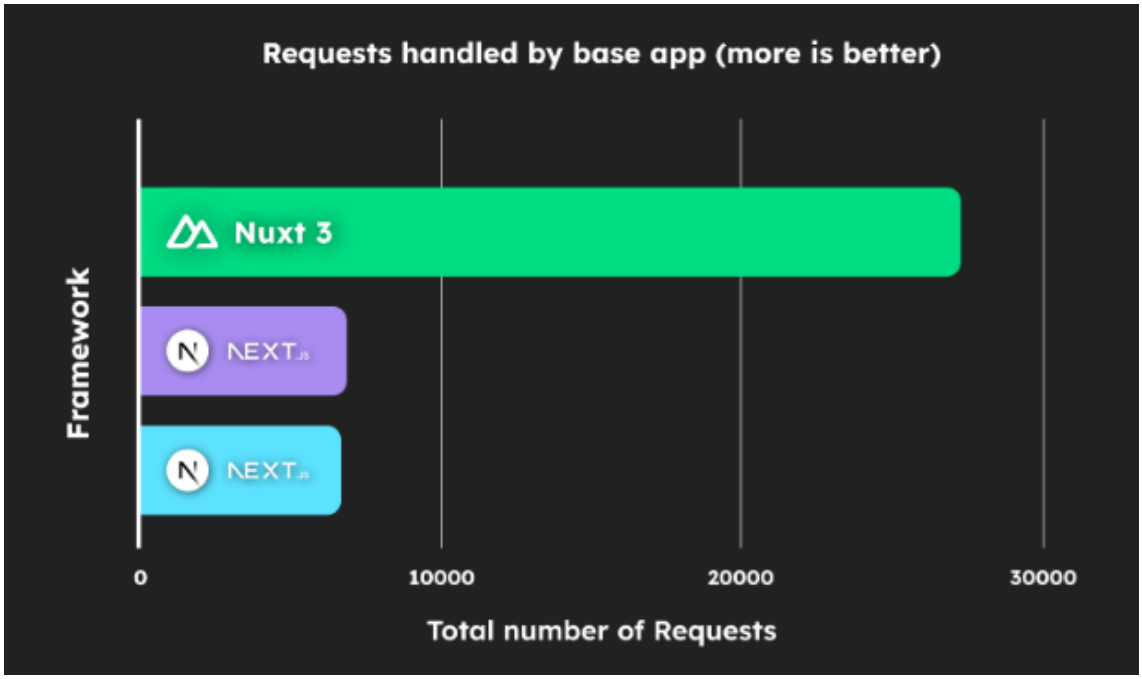
(icarusgk, repo Github « fw-bench »)

Figure 17 : comparatif sur le temps moyen de réponse entre Nuxt et Next en fonction du nombre de tags affichés



(icarusgk, repo Github « fw-bench »)

Figure 18 : comparatif du nombre de requêtes gérées par Nuxt et Next sur le Hello World



(icarusgk, repo Github « fw-bench »)

On peut voir dans les résultats que Nuxt est très performant par rapport à Next, autant en nombre de requêtes par seconde qu'en temps moyen de réponse.

Il est intéressant de voir que le nombre de requêtes par secondes absorbées sur les deux frameworks n'est pas le même. Bien sûr, aucun des deux développeurs n'expliquent cette différence car les deux benchmarks n'ont pas été fait ensemble pour les comparer.

L'une des causes probables pourrait être la configuration physique de l'ordinateur ou du système d'exploitation. Néanmoins, le résultat est assez parlant pour montrer que Nuxt est bien plus performant que Next.

7. Développement de l'application

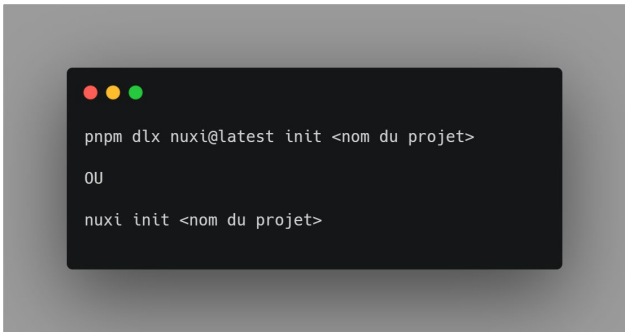
7.1 Comment créer un projet Nuxt

À l'instar des autres aspects faciles à utiliser que j'ai déjà pu vous présenter, l'installation d'un projet Nuxt est très simple.

- Les prérequis, que vous pouvez trouver sur le site de Nuxt[34], sont au nombre de trois : une version de Node.js au moins égale ou supérieure à la 16.10 ;
- Un éditeur de texte (un IDE fera plus l'affaire) ;
- Un terminal pour exécuter les quelques commandes nécessaires.

La première étape est celle de l'initialisation du projet. Vous avez deux possibilités pour le faire : ou vous utilisez npm (ou pnpm), ou bien vous pouvez utiliser, après l'avoir installé (npm install -g nuxi), Nuxi, dont j'ai fait une mention honorable précédemment.

Figure 19 : commande pour installer un projet Nuxt



(Evan Schleret)

Après ça, il faudra vous rendre dans le dossier de votre nouveau projet et installer les dépendances avec la commande « pnpm / npm install ».

Pour démarrer en local votre application, utilisez la commande « pnpm run dev ».

7.2 Concept et fonctionnalités de l'application

Le développement propre de mon application n'étant pas le but principal de mon travail de Bachelor, j'ai décidé de développer une application ayant pour rôle de démonstrateur général des différentes fonctionnalités que proposent Nuxt. Il faut que mon application fasse donc la démonstration de plusieurs concepts clefs de Nuxt que j'ai pu vous présenter jusqu'ici, comme l'accès et l'interaction à une API dans le but de récupérer ou envoyer de la donnée, utiliser le principe d'auto-importation, le routage dynamique et différents modules, tel que celui servant à l'authentification. D'autres concepts peuvent être utilisés.

[34] https://nuxt.com/docs/getting-started/installation

En tenant compte de ce qui précède, j'ai décidé de développer une application de blogging interactive. Cette application serait pourvue des fonctionnalités suivantes :

- Authentification : un système d'inscription et de connexion est présent et permet à des utilisateurs d'avoir leur propre compte.
- Création d'articles : les utilisateurs authentifiés peuvent rédiger et publier des articles de blog. Ils ont la possibilité d'intégrer du texte, des images et des vidéos (pas directement hébergées sur l'application).
- Partage et collaboration : les utilisateurs authentifiés peuvent partager leurs articles sur leur page personnelle et donc les montrer à la communauté de l'application.
- Recherche et filtrage : les utilisateurs de l'application peuvent chercher des articles en fonction de leur titre ou selon son auteur.
- Fonctionnalité « bonus » : les utilisateurs authentifiés peuvent commenter les articles des autres utilisateurs.

7.3 Commentaires et explications sur certaines parties du développement et du code de l'application

Le début du développement de mon application débute avec l'initialisation d'un nouveau projet Nuxt. Pour rappel, vous trouvez la marche à suivre pour l'installation d'un nouveau projet au chapitre Comment créer un projet Nuxt.

La partie design de l'application n'étant vraiment pas importante, j'ai utilisé un template qui s'appelle Tairo[35] et que j'ai acheté pour l'occasion sur le site cssninja.io[36]. Le thème est par ailleurs mis en avant par Nuxt sur leur site.

La partie suivante a été de mettre en place les différentes pages de base : la page d'authentification ou de création de son compte.

Justement, pour l'authentification, à ce jour (mercredi 12 juillet 2023), Nuxt n'a pas encore sorti de module officiel prenant en charge toute cette partie liée au développement d'applications web. Néanmoins, il propose et met en avant un module de sidebase nommé nuxt-auth[37], qui répond entièrement aux scénarios de base, c'est-à-dire une authentification basée sur la paire email / mot de passe, ou des besoins plus grands comme l'authentification avec des services externes tels que Github, Google et d'autres.

Commençons concernant les pages avec la définition des données d'enregistrement d'un membre. Seront à renseigner un nom, une adresse email et un mot de passe. Optionnellement, l'utilisateur aura le choix d'utiliser un avatar. Ce sont les données de

[35] https://cssninja.io/product/tairo
[36] https://cssninja.io/
[37] https://github.com/sidebase/nuxt-auth

base. Bien sûr, d'autres informations tels que les réseaux sociaux de l'utilisateur pourront être renseignés dans son profil.

Les deux pages nécessiteront donc des validations de champ et Nuxt propose aussi dans ce cas un module nommé Vee-Validate[38]. Couplé à la librairie zod[39], vous aurez alors toutes les cartes en main pour faire de la validation de champ.

Lors de l'élaboration de la page d'enregistrement, j'ai mis quelques bonnes dizaines de minutes à me rendre compte d'un léger problème… Je disais, il y a à peine deux paragraphes, que je souhaitais que l'utilisateur créant son compte puisse définir un avatar, étant donc finalement l'envoi d'un fichier à mon API. Or, les données étant envoyées à ladite API au format JSON, j'ai mis un moment à me rendre compte du problème auquel je me heurtais et donc pourquoi est-ce que je ne retrouvais pas la trace de l'avatar dans la requête faite à l'API.

Après m'être creusé passablement la tête, deux idées me sont venues pour palier à ce problème :

- La première possibilité, si je souhaitais complètement garder ma structure JSON, serait d'encoder l'avatar / le fichier en base64 pour le transmettre alors dans la structure transmise à l'API ;

  Cette option est satisfaisante du point de vue de mon besoin, à savoir transmettre un JSON, mais il présente deux problématiques parfaitement liées : le fait de devoir encoder à décoder le fichier en base64 pourra amoindrir les performances de l'application dû à l'opération d'encodage / décodage.

  De plus, comme il en est fait mention dans RFC 2045 de l'ietf[40], la surcharge occasionnée par l'encodage d'un fichier en base64 est généralement de 33%, ce qui ne manquera alors pas d'augmenter la taille de la requête faite à l'API.

  Le dernier problème que j'identifie avec cette option est qu'elle ne fonctionne vraiment que si l'on envoie qu'un seul fichier. En effet, le fait d'en envoyer plusieurs fichiers signifie devoir surcharger notre JSON également, en effectuant par ailleurs plusieurs fois l'instruction d'encodage.

- La seconde option serait d'envoyer l'image en un second temps en utilisant le même format de donnée qu'est créé en JavaScript lorsqu'un formulaire HTML est validé : l'objet FormData.

  Il permet alors d'ajouter le fichier et de l'envoyer à l'API, sans format JSON cette fois.

  L'envoi se ferait suite à la confirmation par l'API que l'objet initial a bien été créé et persisté en base de données. L'API renverrait à Nuxt l'ID qui s'en servirait donc, au travers d'une seconde requête, d'envoyer le fichier avec l'ID pour lier le fichier à l'objet.

[38] https://vee-validate.logaretm.com/v4/
[39] https://zod.dev/
[40] https://www.ietf.org/rfc/rfc2045.txt

Souhaitant avancer sur le développement propre de l'application, j'ai décidé de ne pas m'attarder sur ce point trop longtemps. Je dois cependant reconnaître que je n'y avais pas pensé et il semblerait que ni Nuxt ni Vue n'aient trouvé de parade à ce cas d'usage lors du développement d'une application.

Une fois les pages d'enregistrement et de connexion terminées, je me suis mis poser sur papier les différentes pages qui concerneraient un article de blog. Il y en aurait quatre : une qui gérerait la page d'accueil, qui contiendrait une liste des dix derniers blogs postés, avec une possibilité de pagination au bas de la page, pour voir les anciens blogs, une autre pour afficher un blog particulier, une pour l'éditer et bien sûr, une pour en créer.

J'ai d'abord commencé à créer la page permettant l'ajout de nouveau blog. Les champs présents dans le formulaire de la page sont au nombre de quatre : un titre, une image de couverture, un éditeur de texte enrichi et une liste déroulante permettant de choisir « l'état » d'affichage du blog (brouillon, privé ou publié). Pour l'éditeur de texte enrichi, j'ai choisi celui de tiny.cloud[41]. Il a plusieurs intégrations différentes, dont une pour Vue.

Je ne l'ai pas clairement abordé dans le point en parlant mais Nuxt vous permet d'ajouter des dépendances faites pour Vue même quand elles n'ont alors pas du tout été prévues pour être ajoutées à Nuxt. Dans le cas de la librairie de tiny.cloud, l'installation s'est simplement faite via l'ajout d'un paquet avec pnpm. Ensuite, il a fallu important le composant dans Nuxt et cela s'est fait via la notion de plugins.

Comme expliqué sur la page de la documentation y afférant[42], Nuxt va automatiquement lire les plugins que vous écrirez ou ajouterez dans le dossier « plugins » puis les charger lors de la création de l'instance de Vue dans votre application.

Nuxt vous permet également d'ajouter un suffixe au nom du plugin permettant de dire s'il n'est fait que pour le côté client ou que pour le côté serveur. Dans notre cas, le plugin de sera utilisé que du côté client. Le nom de mon plugin (le nom du fichier) est alors tinymce.client.ts.

L'ajout du composant d'éditeur de TinyCloud se fait tout aussi facilement que pour installer la dépendance ; il suffit d'ajouter les quatre lignes suivantes :

[41] https://www.tiny.cloud/
[42] https://nuxt.com/docs/guide/directory-structure/plugins

Figure 20 : importation d'un composant fait pour Vue dans une application Nuxt

(Evan Schleret)

nuxtApp est l'instance de votre application Nuxt. vueApp est celle de Vue. Le premier argument passé dans « component » est le nom du composant tel qu'on souhaite pouvoir l'utiliser dans notre application et le second argument est le lien au véritable composant que l'on souhaite importer.

Une fois Nuxt relancé, celui-ci lit le fichier du plugin, le traite et l'ajoute à l'application. J'ai alors pu l'utiliser en y faisant appel via le composant « Editor ».

Figure 21 : appel du composant dans une page de l'applciation

(Evan Schleret)

Le composant maintenant importé via son plugin et mis sur ma page, je pouvais alors l'utiliser comme un véritable éditeur de texte enrichi. Il permettrait aux utilisateurs de mon blog de pouvoir insérer du texte avec de la stylisation, ajouter des images externes, faire de la mise en page, etc. L'aspect pratique est que la valeur récupérée est une simple chaîne de caractères contenant le code HTML généré par l'éditeur.

Comme pour la problématique liée à l'envoi de l'avatar, je n'ai pas souhaité perdre du temps au début du développement de ma plateforme de blog en essayant de rendre l'éditeur capable d'accepter le glisser-déposer ou le copier-coller d'image dynamique, l'éditeur permettant déjà pour l'instant d'ajouter des images en mettant l'URL de ladite image.

Pour faire des appels à une API, je vous ai présenté les composants useFetch, useLazyFetch, useAsyncData et useLazyAsyncData. Lors du développement de mon application, pour éviter de faire de la duplication de code en devant systématiquement spécifier des informations qui se répètent, comme l'URL de base de mes requêtes et les deux entêtes spécifiant le type de valeur de retour attendue et l'authentification via mon jeton d'API, j'ai créé mes propres composables useApiFetch et useApiLazyFetch.

Figure 22 : composable personnalisé useApiFetch

```
// composables/useApiFetch.ts
import {useFetch} from "#app/composables/fetch";
import {useAuth} from "#imports";

export const useApiFetch = (request: string, opts?: any) => {
    const config = useRuntimeConfig()
    const { token } = useAuth()

    return useFetch(request, {
        baseURL: config.public.apiBaseUrl + '/',
        headers: {
            Authorization: token.value,
            Accept: 'application/json'
        },
        ...opts
    })
}
```

(Evan Schleret)

De cette façon, je peux maintenant faire des appels à mon API, protégées des appels non authentifiés, sans devoir spécifier plusieurs fois les mêmes informations de base. Vous observez que je peux ajouter autant d'options supplémentaires en appelant ma méthode car j'ai spécifier un argument optionnel « opts?: », qui représente un tableau, que je passe à la fonction useFetch.

La version useApiLazyFetch est identique à useApiFetch, à la différence que le paramètre « lazy : true » est passé dans la fonction.

Une fois la page de création terminée, il a fallu créer celle qui afficherait le blog créé. Les éléments particuliers de cette page sont au nombre de trois : l'affichage d'un bouton pour éditer le blog devra être conditionné au fait qu'il appartienne à l'utilisateur connecté, un espace pour commenter le blog doit être présent au bas de la page et les dates retournées par l'API doivent être lues dans un format digeste pour les humains. En effet, l'API ne retourne pas une date au format « 21.07.2023 à 18h35 » mais une chaîne de

caractères au format ISO 8601[43]. Le format de la date et l'heure que je viens d'écrire sera alors retourné par l'API comme « 2023-07-21T18:35:00+0200 ». Il y a mieux en termes de lisibilité…

Pour résoudre le point que je viens d'aborder, j'ai fait appel à la librairie VueUse[44]. Elle offre une multitude de méthodes répondant à des besoins variés. Celui dont j'ai eu besoin était donc lié au temps. Cette librairie offre les méthodes « useDateFormat » et « useTimeAgo ». Elles permettent respectivement de formater un timestamp d'afficher le temps écoulé depuis un timestamp.

Figure 23 : exemple d'utilisation de la métohode useDateFormat

(Documentation sur le site de VueUse)

L'affichage conditionnel du bouton pour éditer le blog était simple à faire. Il fallait simplement comparer le UUID de l'utilisateur connecté à celui de l'utilisateur ayant écrit le billet de blog. Pour cela, j'ai simplement utilisé la fonctionnalité de vue permettant de faire des conditions sur des éléments HTML (« v-if=condition »).

[43] https://en.wikipedia.org/wiki/ISO_8601
[44] https://vueuse.org/

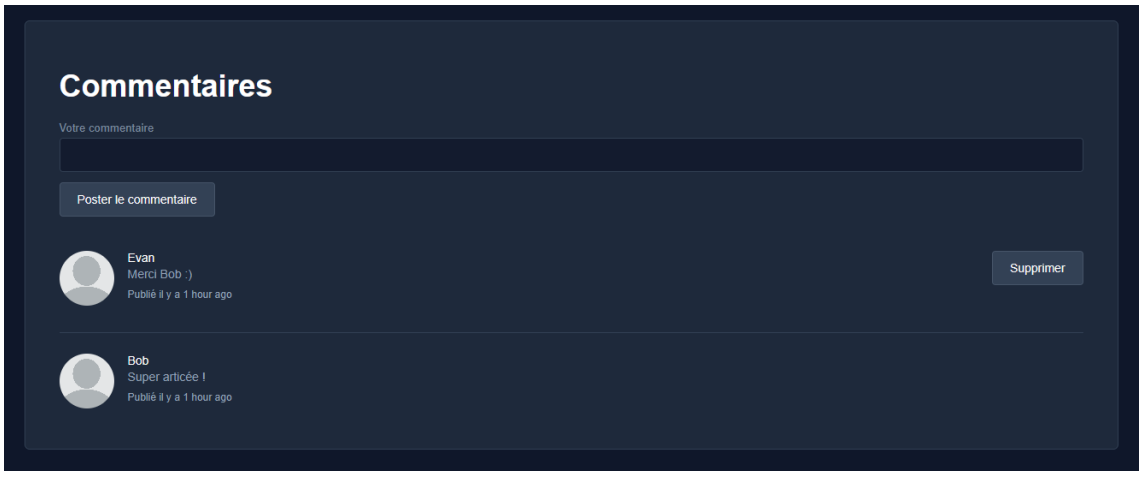
Figure 24: affichage conditionnel du bouton d'édition du billet de blog

(Evan Schleret)

Je vous montrerai après que le simple fait de mettre un affichage conditionnel ne suffit bien évidemment pas pour interdire à un utilisateur d'éditer un billet de blog dont il ne serait pas l'auteur.

La partie la plus intéressante de la page, la dernière, est la partie permettant d'ajouter des commentaires sur les billets de blog. Pour la faire, j'ai ajouté une section au bas de la page et j'y ai mis un formulaire permettant d'ajouter un commentaire. Pour l'affichage desdits commentaires, j'ai créé un composant, qui aurait également un affichage conditionnel pour le bouton de suppression du commentaire. L'utilisation de la méthode useTimeAgo s'est par ailleurs faite dans ce composant.

Figure 25 : sections des commentaires avec l'affichage conditionnel du bouton « supprimer »



(Evan Schleret)

Mon composant Comment est court en termes de nombre de lignes de code mais il embarque plusieurs des concepts que je vous ai montré jusqu'ici. Il y a notamment une

partie du code qui utilise le TypeScript, j'y utilise une librairie (déjà installée avec le thème que j'utilise) pour l'affichage de notification mais également celle gérant l'aspect de la session des utilisateurs de mon application. Un des autres concepts que j'utilise, qui vient de Vue et que je n'ai pas spécifiquement présenté est la définition d'évènement que l'on peut émettre et transmettre au composant parent.

Figure 26 : exemple de composant envoyant un évènement à son parent

```vue
// components/ChildComponent.vue
<script setup lang="ts">
const emit = defineEmits<{
  'my-custom-event'
}>()

const triggerFunction = () => {
  emit('my-custom-event', <arguments optionnels...>)
}
</script>

<template>
    <div>
        <button @click="triggerFunction">
            Appeler ma fonction qui émet un évènement
        </button>
    </div>
</template>

// pages/index.vue
<template>
    <ChildComponent @myCustomEvent="(valeurs optionnelles)" /> // ou @my-custom-event=""
</template>
```

(Evan Schleret)

Figure 27 : composant Comment de mon application

```vue
<script setup lang="ts">
import {Comment} from "~/types/Comment";

const props = defineProps<{
    comment: Comment,
    index: any
}>()

const route = useRoute()
const toaster = useToaster()
const {data: user} = useAuth()
const deletePending = ref(false)
const emit = defineEmits<{
    'delete'
}>()

const deleteComment = async () => {
    deletePending.value = true
    toaster.clearAll()

    try {
        const {data: response, error} = await useApiFetch('blogs/' + route.params.blog + '/comments/' +
props.comment.uuid, {
            method: 'DELETE',
        })

        if (error.value?.statusCode === 403) {
            await toaster.show({
                title: 'Oops !',
                message: 'Ce commentaire ne vous appartient pas !',
                color: 'danger',
                icon: 'lucide:alert-triangle',
                closable: true,
            })
            return
        }

        if (response.value.status === 'success') {
            await toaster.show({
                title: 'Commentaire supprimé',
                message: `Votre commentaire a été supprimé`,
                color: 'success',
                icon: 'ph:check',
                closable: true,
            })

            emit('delete')
        }
    } catch (errors: any) {
        await toaster.show({
            title: 'Oops !',
            message: 'Votre commentaire n\'a pas pu être supprimé. Réessayez plus tard.',
            color: 'danger',
            icon: 'lucide:alert-triangle',
            closable: true,
        })
    }

    deletePending.value = false
}
</script>

<template>
    <div>
        <div
            class="flex flex-col gap-4 sm:flex-row"
            :class="index > 0 ? 'pt-6' : ''"
        >
            <NuxtLink :to="'/-/users/' + comment.user?.uuid">
                <BaseAvatar
                    size="lg"
                    :src="comment.user?.profile_picture_path"
                    :text="comment.user?.name"
                    :data-tooltip="comment.user?.name"
                />
            </NuxtLink>

            <div class="max-w-md">
                <BaseHeading
                    as="h3"
                    size="sm"
                    weight="light"
                    lead="tight"
                    class="text-muted-800 dark:text-white"
                >
                    <span>
                        {{ comment.user.name }}
                    </span>
                </BaseHeading>
                <BaseParagraph
                    size="sm"
                    class="text-muted-500 dark:text-muted-400 mb-1"
                >
                    <span>{{ comment.content }}</span>
                </BaseParagraph>
                <BaseParagraph size="xs" class="text-muted-400">
                    <span>Publié il y a {{ useTimeAgo(comment.created_at, {
                        updateInterval: 10,
                    }).value }}</span>
                </BaseParagraph>
            </div>
            <div class="w-full sm:ms-auto sm:w-auto" v-if="comment.user?.uuid === user?.data.uuid">
                <BaseButton color="default" class="w-full sm:w-auto" :loading="deletePending" @click="deleteComment">
                    <span>Supprimer</span>
                </BaseButton>
            </div>
        </div>
    </div>
</template>
```

(Evan Schleret)

Figure 28 : intégration du composant Comment dans une page

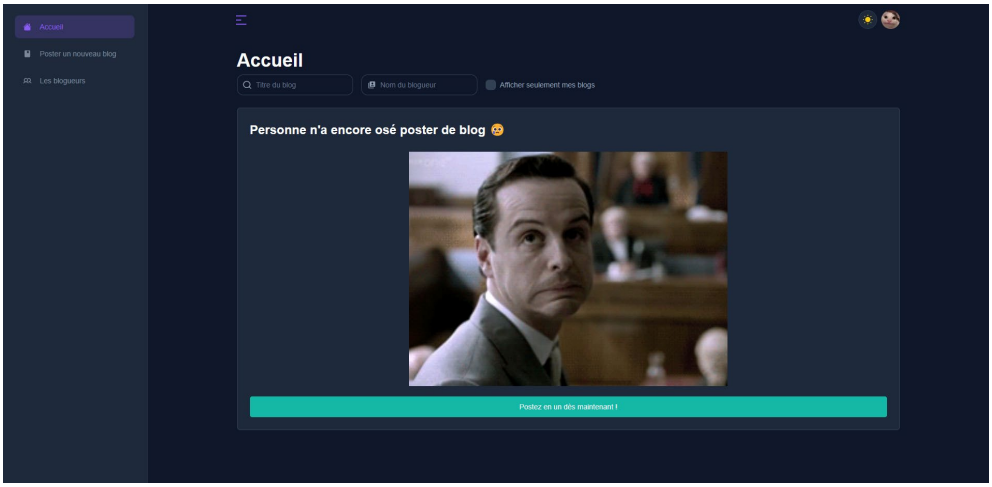
```vue
// pages/~/blogs/[blog]/index.vue
<script setup lang="ts">
const route = useRoute()
const {data: comments, refresh, pending} = await useApiFetch('blogs/' + route.params.blog + '/comments')
</script>

<template>
    <div v-if="comments.data.data">
        <Comment
            :index="i"
            :comment="comment"
            :key="comment.id"
            v-for="(comment, i) in comments.data.data"
            @delete="refresh"
        />
    </div>
</template>
```

(Evan Schleret)

Après avoir terminé la page d'affichage d'un blog, j'ai terminé avec la page affichant l'ensemble des blogs. J'y ai prévu le fait que l'utilisateur connecté puisse filtrer les blogs selon trois aspects, augmentant sa probabilité de trouver ce qu'il cherche : le titre du blog, son auteur (il devra y avoir de l'auto-complétion sur tous les utilisateurs du blog) et une case pour n'afficher que ses propres blogs.

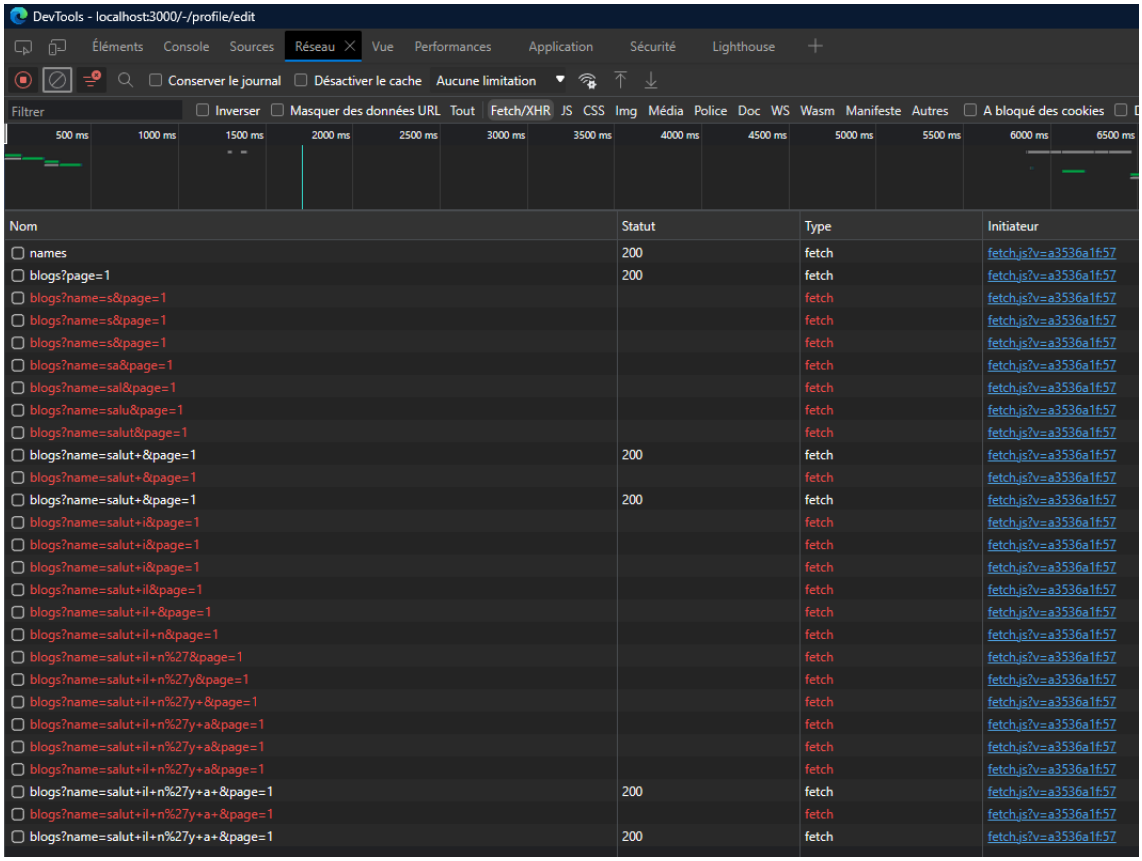
Figure 29 : affichage de la page d'accueil et de ses filtres



(Evan Schleret)

Pour faire ces filtres, j'ai du trouvé un moyen de ne pas surcharger mon API. En effet, par défaut, quand on utilise useFetch et qu'on lui passe en paramètres le fait de devoir surveiller un composant Vue réactif, il n'y a pas d'attente entre le moment où on le met à jour et le moment où useFetch va aller « reprendre » les données sur l'URL spécifiée.

Figure 30 : affichage des requêtes effectuées quand on met à jour un composant réactif de Vue et que useFetch le surveille



(Evan Schleret)

Vous pouvez voir sur la figure ci-dessus le problème : il y a énormément de requêtes qui sont effectuées et cela peut rapidement surcharger notre API, en finissant probablement par nous bloquer en raison d'un nombre trop important de requêtes (erreur 429 Too Many Requests[45]).

Pour corriger ce problème, j'ai du d'abord faire en sorte que les paramètres qui seraient mis à jour dans mon URL ne se mettraient à jour qu'à des intervalles définis, par exemple chaque seconde. Pour ça, VueUse propose la méthode watchThrottled, qui prend en paramètres un ou plusieurs composants Vue réactifs, une fonction callback qui va exécuter le code souhaité et enfin les options de la méthode, notamment tous les combien de temps elle s'exécute si les valeurs réactives sont constamment mises à jour.

En plus de cette méthode, il faut spécifier à notre fonction qui récupère les données un paramètre qui dit à Nuxt de ne pas surveiller les composants réactifs. Pour ce faire, il faut simplement spécifier le paramètre « watch : false ».

[45] https://developer.mozilla.org/fr/docs/Web/HTTP/Status/429

Avec ces changements, les valeurs que nous entrons dans les composants filtrant nos données ont un décalage d'une seconde à chaque fois qu'un appel à l'API est fait.

Figure 31 : page d'accueil avec les filtres

```vue
// pages/~/index.vue
<script setup lang="ts">
const route = useRoute()
const router = useRouter()
const name = ref(route.query.name)
const author = ref(route.query.author)
const page = computed(() => parseInt((route.query.page as string) ?? '1'))
const onlyMine = ref(route.query.mine)

watchThrottled([name, author, page, onlyMine], async () => {
    await router.push({
        path: '/~/',
        query: {
            name: name.value,
            author: author.value,
            page: page.value,
            mine: onlyMine.value,
        },
        replace: true
    })
    await refresh()
}, { throttle: 1000 })

const { data: blogs, pending, refresh, error } = await useApiLazyFetch<Blog[], Boolean>('blogs', {
    params: {
        name: name,
        author: author,
        page: page,
        mine: onlyMine
    },
    watch: false,
})
const {data: authors} = await useApiFetch('users/names')
</script>

<template>
    <div>
        <BaseInput
            v-model="name"
            icon="lucide:search"
            shape="curved"
            placeholder="Titre du blog"
            :classes="{ wrapper: 'w-full sm:w-auto'}"
        />
        <BaseAutocomplete
            v-model="author"
            icon="material-symbols:switch-account-rounded"
            shape="curved"
            placeholder="Nom du blogueur"
            :classes="{ wrapper: 'w-full sm:w-auto'}"
            :items="authors"
            :i18n="{
                pending: 'Chargement...',
                empty: 'Aucun résultat trouvé.'
            }"
            clearable
        />
        <BaseCheckbox v-model="onlyMine" :true-value="1" :false-value="0" label="Afficher seulement mes blogs" shape="curved" />
    </div>
</template>
```

(Evan Schleret)

L'auto-complétion est faite automatiquement par le composant BaseAutocomplete, fourni par le thème que j'ai utilisé.

Revenons maintenant à la page d'édition de blog. Je vous ai dit que le simple fait de mettre un affichage conditionnel sur le bouton pour éditer le blog ne suffirait pas. En effet, rien n'interdit à un utilisateur mal intentionné de devenir l'URL permettant de modifier ledit blog et s'y rendre en tapant manuellement l'adresse correcte dans sa barre d'URL. Pour interdire l'accès à cette page à un utilisateur non autorisé, j'ai dû utiliser un autre des concepts de Nuxt : les middlewares.

Comme nous l'avons vu, un middleware permet de filtrer une requête et d'y exécuter une logique. Dans notre cas, il faut donc un middleware permettant de vérifier si le blog que nous souhaitons éditer appartient bien à l'utilisateur qui est connecté. Pour cela, rien de très compliqué puisque nous pouvons aisément récupérer les deux informations. La première est contenue dans l'URL (UUID du blog) et l'autre peut être récupérée avec notre bibliothèque nuxt-auth, qui gère la session. La logique va simplement être de comparer si le UUID de l'auteur du blog est le même que le UUID de l'utilisateur connecté. Si ce n'est pas le cas, on définit un message d'erreur qui sera envoyé à l'utilisateur et on le redirige sur la page du blog.

Figure 32 : middleware permettant de vérifier si l'auteur du blog est le même que celui qui est connecté

```
// middleware/blogEdit.ts

export default defineNuxtRouteMiddleware(async (to, from) => {
    const {data: user} = useAuth()
    const {data: blog} = await useApiFetch('blogs/' + to.params.blog)

    // @ts-ignore
    if (blog.value?.data.user.uuid !== user.value?.data.uuid) {
        const toaster = useToaster()

        await toaster.show({
            title: 'Oops !',
            message: 'Vous n\'avez pas la permission d\'éditer ce blog !',
            color: 'danger',
            icon: 'lucide:alert-triangle',
            closable: true,
        })

        return navigateTo('/-/blogs/' + to.params.blog)
    }
})
```

(Evan Schleret)

Maintenant que les fonctionnalités principales de mon application sont faites, il fallait que je revienne sur la fonctionnalité permettant d'envoyer un fichier à mon API, par exemple lors de la création du compte. J'ai choisi la seconde des deux options que j'exposais plus tôt, celle qui consisterait donc à envoyer dans un second temps mon fichier à l'API.

Figure 33 : adaptation de la page de création de compte pour prendre en compte l'ajout d'une image de profil

```
// pages/auth/sign-up.vue
...

const {signIn, data: user} = useAuth()
const {data: response, error} = await
useFetch(useRuntimeConfig().public.apiBaseUrl + 'auth/sign-up', {
    method: 'POST',
    body: {
        email: values.email,
        name: values.name,
        password: values.password,
        password_confirmation: values.password_confirmation,
    }
})
await signIn({
    email: values.email,
    password: values.password,
}, {
    redirect: false,
})
if (response.value.status === 'success' && values.profile_picture) {
    let formData = new FormData()
    formData.append('profile_picture', values.profile_picture?.[0])
    await useApiFetch('users/' + user.value?.data.uuid + '/profile-picture', {
        method: 'POST',
        body: formData
    })
}

...
```

(Evan Schleret)

Comme vous pouvez le voir, trois requêtes à l'API sont exécutées : une première pour créer le compte, une autre pour authentifier le compte sur l'application du point de vue de nuxt-auth et une troisième pour envoyer, si l'utilisateur en a fourni une, la photo de profil.

J'ai fait de même pour la création de blog, en permettant à l'utilisateur d'ajouter une photo de couverture.

Conclusion

Me voilà donc à l'issue de ce travail de Bachelor, épreuve ultime avant l'obtention du précieux sésame. Ce travail a été très intéressant à faire. D'abord car j'aime beaucoup les technologies liées au développement web, ensuite car j'ai eu l'occasion d'apprendre beaucoup de choses à mesure que j'écrivais ce travail. J'ai d'ailleurs dû plusieurs fois relire certains concepts que j'expliquais car j'étais ou trop flou ou pas vraiment dans le concept.

Mon objectif principal était d'analyser le framework Nuxt, sous sa nouvelle version majeure, et d'essayer d'expliquer, le plus objectivement possible, pourquoi il est un candidat idéal lorsqu'un développeur ou son équipe souhaite se lancer dans le développement d'une nouvelle application, quelque puisse être sa taille ou les besoins.

Nous avons en effet vu qu'au travers des nombreuses fonctionnalités, méthodes, fonctions, API, etc. mises à disposition des développeurs, Nuxt coche un vaste ensemble de cases, dans bien des domaines, donc voici quelques exemples parmi la liste non exhaustive : les performances, le SEO, la simplicité d'apprentissage et d'utilisation, le système de routage dynamique, le temps gagné, etc.

La première problématique à laquelle j'ai été confronté a été celle de la connaissance même du framework. En effet, si je n'étais pas étranger à l'usage des frameworks, ayant déjà développé moultes applications au cours des trois dernières années avec le framework PHP Laravel, je n'avais jamais vraiment travaillé avec JavaScript, ayant jusqu'ici préféré le PHP au JavaScript, langage avec lequel j'étais moins à l'aise, ayant préféré développer côté backend. Avec l'essor des frameworks JavaScript, j'ai décidé de m'y intéresser. De plus, avec le PHP, bien qu'on puisse faire énormément de choses avec, en témoigne son utilisation par de grands sites internet pour leur backend, tels que Yahoo, Etsy, Wikipedia ou Fandom[46], on se heurte assez vite à un plafond de verre, PHP étant assez limité lorsqu'on souhaite intégrer du vrai dynamisme comme JavaScript peut le faire.

En regardant du côté de Laravel, je suis tombé sur Inertia.js[47]. Néanmoins, avec le temps, j'ai remarqué que ce choix n'était vraiment pas la solution idéale si je souhaitais collaborer avec d'autres personnes. En effet, le fait de réunir à la fois son backend et son frontend dans le même projet peut rapidement provoquer le fait qu'en étant plusieurs à travailler sur l'application, même avec des outils de versionnage comme Git, on perd

[46] https://en.wikipedia.org/wiki/Programming_languages_used_in_most_popular_websites
[47] https://inertiajs.com/

du temps à résoudre les conflits. De plus, cela rend l'application compliquée à utiliser dans des environnements où il faudrait répliquer ou redonder l'application.

C'est ainsi que je m'étais remis dans la recherche, bien que non active, d'un nouveau framework JavaScript. Si Vue.js, après l'avoir utilisé au travers de son intégration via Inertia.js, semblait être le candidat parfait, le hasard des choses a fait que, dans le courant du mois de septembre 2022, une vidéo a attiré mon regard sur mon accueil YouTube. Il s'agissait d'une présentation de la part d'un youtubeur parlant de tech, Fireship[48], présentant un tout nouveau framework à venir… Nuxt.js 3.

Dès la fin de la vidéo, je me suis empressé d'aller sur le site de Nuxt et j'ai commencé à m'intéresser au framework. Rapidement, j'ai également commencé à l'utiliser, bien qu'à ce moment-là, il était encore en pleine phase de développement, sans aucune annonce officielle ne donnant sa date de sortie officielle, mais cela ne semblait être pas trop loin, étant déjà en release candidates. De nombreux bugs étaient présents, mais le framework semblait vraiment avoir beaucoup de potentiel…

Le 4 novembre 2022, le compte officiel de Nuxt annonçait la sortie de la 13 release candidate en mentionnant le fait qu'il n'y aurait « plus qu'une ou deux release candidates » avant la version finale et la sortie officielle de Nuxt 3.0.[49] et c'est ainsi que Nuxt 3.0 était enfin sorti le 16 novembre 2022[50].

J'ai commencé à refaire certains de mes petits sites avec Nuxt, maintenant qu'il était sorti en version 3 stable et c'est tout naturellement que quand il a fallu trouver un sujet pour mon travail de Bachelor, je me suis dit que le faire sur Nuxt serait une bonne idée, tenant compte de mon niveau encore assez basique et en raison de l'aspect encore frais donc peu étudié sur le sujet.

L'évocation de ce dernier point me donne une transition toute trouvée à un second problème que j'ai quand même eu lors de l'écriture de mon travail, mais également lors du développement de mon application : Nuxt.js 3 est nouveau.

Alors que quand on cherche des problèmes qu'on pourrait rencontrer sur Vue 2, React ou d'autres frameworks bien utilisés et dont les versions existent depuis un moment, Nuxt 3 peut parfois souffrir, toute proportion gardée, de sa nouveauté. Bien que la communauté soit déjà grande et qu'elle continue de grandir, on ne trouve pas forcément

[48] https://www.youtube.com/@Fireship
[49] https://twitter.com/nuxt_js/status/1588535895883272192?s=20
[50] https://twitter.com/nuxt_js/status/1592904778337906689

du premier coup la réponse à sa question si on a un souci ou que l'on souhaite en apprendre davantage sur un concept.

Si je devais maintenant répondre à la question portant sur ma problématique, à savoir « est-ce que Nuxt peut être un bon candidat pour développer des applications web », je répondrai par l'affirmative, en nuançant un peu ma réponse.

Nuxt est d'abord un très bon candidat en raison de sa courbe d'apprentissage : n'ayant eu que pour base l'utilisation de Vue au travers d'Inertia.js dans mes applications Laravel, je n'avais jamais vraiment fait de vraie application purement en JavaScript. Néanmoins, cela n'a pas été un souci avec Nuxt car comme avec Laravel, il vient avec tellement d'éléments simplifiés dans leur usage que cela en devient beaucoup plus facile. Repensez à l'analogie avec la boîte de Lego que je faisais au début de mon travail de bachelor, en présentant les frameworks. Nuxt en est très bon exemple. Pour beaucoup de concepts, tels que le routage, la mise en cache, le partage d'état, beaucoup de choses viennent « out-of-the-box » et ne nécessitent ni configuration particulière, ni de retourner dans tous les sens son code pour le faire fonctionner.

Ensuite, comme je vous l'ai présenté et répété, Nuxt peut être utilisé pour tout type d'application web, que vous souhaitiez faire un site institutionnel, un réseau social, un site d'e-commerce ou même un site de streaming. Plusieurs entreprises font d'ailleurs confiance à Nuxt pour leur site. On peut notamment citer OpenAI, Ecosia, le site d'Adobe pour les fonts, Volta.net, TikTok ou encore le site internet du Jet Propulsation Laboratory de la Nasa[51].

Je disais que je répondrai à la question en nuançant et c'est surtout autour de l'aspect de la nouveauté du framework, que j'ai déjà évoqué juste avant, que la nuance tourne. En effet, bien qu'étant aujourd'hui dans une version stable et que quelques mises à jour soient déjà sorties pour encore apporter du nouveau contenu au framework, Nuxt manque encore de réponses déjà données à des questions posées sur des forums comme StackOverflow. De plus, les intégrations natives pour Nuxt ne sont pas encore abondantes, bien que nous ayons vu que beaucoup de bibliothèques pour Vue.js 3 sont installables avec Nuxt.

Si je devais maintenant poser quelques éléments pour de futurs travaux de Bachelor sur le même sujet, je dirais qu'il faudrait peut-être plus appuyer sur l'étude plus poussée des performances d'une application identique mais faite avec plus de frameworks.

[51] https://nuxt.com/showcase

L'idée serait de développer une application à l'apparence gourmande, en termes de ressources de calcul, de compilation et / ou interagissant de façon active avec une API, de la déporter sur plusieurs autres frameworks et de les tester individuellement, dans les mêmes conditions, en les poussant au bout de ce qu'elles sont capables de supporter en termes de charge, autant du côté client que serveur.

Une autre idée pour un futur travail de Bachelor serait d'étudier la faisabilité de créer une application Nuxt offrant la possibilité d'ajouter des fonctionnalités supplémentaires à ladite application, en mettant à disposition une API propre à celle-ci. Par exemple, le système de forum XenForo[52] permet ce fonctionnement, en donnant la possibilité aux gérants de forum d'ajouter des plugins, des thèmes, etc.

Pour vraiment conclure sur ce travail de Bachelor, je finirai par dire que je suis satisfait de ce que j'ai écrit. Je pense avoir rempli la mission de base en répondant à la problématique posée, à savoir est-ce qu'un développeur, travaillant seul ou en équipe, devrait choisir Nuxt pour ses futurs projets. En se basant sur mon travail, il pourra en tout cas se faire une idée assez précise de la pertinence du choix de Nuxt pour sa future application.

[52] https://xenforo.com/

Bibliographie

Framework, 2023, *Wikipédia* [en ligne]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=Framework&oldid=203281655 [consulté le 12 avril 2023]. Page Version ID: 203281655

Web framework, 2023, *Wikipedia* [en ligne]. Disponible à l'adresse : https://en.wikipedia.org/w/index.php?title=Web_framework&oldid=1165179472 [consulté le 15 avril 2023]. Page Version ID: 1165179472

Bibliothèque logicielle, 2021, *Wikipédia* [en ligne]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=Biblioth%C3%A8que_logicielle&oldid=184580617 [consulté le 18 avril 2023]. Page Version ID: 184580617

Faker | Faker, [sans date] [en ligne]. Disponible à l'adresse : https://fakerjs.dev/ [consulté le 20 avril 2023 2023].

FakerPHP / Faker, [sans date] [en ligne]. Disponible à l'adresse : https://fakerphp.github.io/None [consulté le 23 avril 2023].

BESSA, Par Anis, 2021. Langages de programmation et Framework - Captiva -. *Captiva* [en ligne]. 10 juillet 2021. Disponible à l'adresse : https://captiva-it.com/blog/tribu-open-source/langages-de-programmation-et-framework/ [consulté le 25 avril 2023].

Component-Based JavaScript Architecture | Sean C Davis, [sans date] [en ligne]. Disponible à l'adresse : https://www.seancdavis.com/posts/component-based-js-architecture/ [consulté le 26 avril 2023].

Understanding Component-Based Architecture | by dshaps | Medium, [sans date] [en ligne]. Disponible à l'adresse : https://medium.com/@dan.shapiro1210/understanding-component-based-architecture-3ff48ec0c238 [consulté le 28 avril 2023].

Programmation déclarative : lorsque l'objectif est plus important que le déroulement, 2020*IONOS Digital Guide* [en ligne]. Disponible à l'adresse : https://www.ionos.fr/digitalguide/sites-internet/developpement-web/programmation-declarative/ [consulté le 2 mai 2023].

LINKS, Ibraci, [sans date]. Qu'est-ce qu'un framework en programmation et pourquoi est-il utile ? | Ibraci Links. [en ligne]. Disponible à l'adresse : https://ibracilinks.com/blog/quest-ce-quun-framework-en-programmation-et-pourquoi-est-il-utile [consulté le 4 mai 2023].

Modèle-vue-contrôleur, 2023*Wikipédia* [en ligne]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=Mod%C3%A8le-vue-contr%C3%B4leur&oldid=205208224 [consulté le 6 mai 2023]. Page Version ID: 205208224

Model–view–viewmodel, 2023*Wikipedia* [en ligne]. Disponible à l'adresse : https://en.wikipedia.org/w/index.php?title=Model%E2%80%93view%E2%80%93viewmodel&oldid=1164384181 [consulté le 9 mai 2023]. Page Version ID: 1164384181

Programmation orientée composant, 2021*Wikipédia* [en ligne]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=Programmation_orient%C3%A9e_composant&oldid=188235726 [consulté le 10 mai 2023]. Page Version ID: 188235726

Jonathan Ive, 2023*Wikipédia* [en ligne]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=Jonathan_Ive&oldid=205506990 [consulté le 11 mai 2023]. Page Version ID: 205506990

Jonathan Ive interview: simplicity isn't simple, [sans date] [en ligne]. Disponible à l'adresse : https://www.telegraph.co.uk/technology/apple/9283706/Jonathan-Ive-interview-simplicity-isnt-simple.html [consulté le 14 mai 2023].

Helmet.js, [sans date] [en ligne]. Disponible à l'adresse : https://helmetjs.github.io/ [consulté le 16 mai 2023].

Component Events | Vue.js, [sans date] [en ligne]. Disponible à l'adresse : https://vuejs.org/guide/components/events.html [consulté le 16 mai 2023].

Node.js, 2023 *Wikipédia* [en ligne]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=Node.js&oldid=206269185 [consulté le 21 mai 2023]. Page Version ID: 206269185

Nuxt.js, 2023 *Wikipédia* [en ligne]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=Nuxt.js&oldid=200318403 [consulté le 22 mai 2023]. Page Version ID: 200318403

Introduction · Get Started with Nuxt, [sans date] [en ligne]. Disponible à l'adresse : https://nuxt.com/docs/getting-started/introduction [consulté le 24 mai 2023].

Hot Module Replacement, [sans date]*webpack* [en ligne]. Disponible à l'adresse : https://webpack.js.org/concepts/hot-module-replacement/ [consulté le 27 mai 2023].

Server-Side Rendering (SSR) | Vue.js, [sans date] [en ligne]. Disponible à l'adresse : https://vuejs.org/guide/scaling-up/ssr.html#what-is-ssr [consulté le 29 mai 2023].

Edge computing, 2023*Wikipedia* [en ligne]. Disponible à l'adresse : https://en.wikipedia.org/w/index.php?title=Edge_computing&oldid=1166708252 [consulté le 30 mai 2023]. Page Version ID: 1166708252

:icarus.gk sur Twitter, 2023 *Twitter* [en ligne]. Disponible à l'adresse : https://twitter.com/icarusgkx/status/1664015769280163840 [consulté le 3 juin 2023].

NANDWANA, Vedant, 2023. *vedantnn71/nuxt-next-perf-test* [logiciel] [en ligne]. 13 juillet 2023. [consulté le 30 juillet 2023]. Disponible à l'adresse : https://github.com/vedantnn71/nuxt-next-perf-test [consulté le 5 juin 2023].

Best Features in Nuxt 3, [sans date] [en ligne]. Disponible à l'adresse : https://masteringnuxt.com/blog/best-features-in-nuxt-3 [consulté le 7 juin 2023].

ALEX, idukpaye, 2022. NuxtJS 3, Let's Talk About 3 New Features. *Medium* [en ligne]. 1 mai 2022. Disponible à l'adresse : https://levelup.gitconnected.com/nuxtjs-3-lets-talk-about-3-new-features-b70fffc5c407 [consulté le 8 juin 2023].

What's new in Nuxt 3?, 2021*DEV Community* [en ligne]. Disponible à l'adresse : https://dev.to/this-is-learning/what-s-new-in-nuxt-3-37lg [consulté le 12 juin 2023].

Rendering Modes · Nuxt Concepts, [sans date]*Nuxt* [en ligne]. Disponible à l'adresse : https://nuxt.com/docs/guide/concepts/rendering#client-side-rendering [consulté le 13 juin 2023].

Auto-imports · Nuxt Concepts, [sans date]*Nuxt* [en ligne]. Disponible à l'adresse : https://nuxt.com/docs/guide/concepts/auto-imports#disable-auto-imports [consulté le 17 juin 2023].

ASAOLU, Elijah, 2021. What's new in Nuxt 3. *LogRocket Blog* [en ligne]. 18 novembre 2021. Disponible à l'adresse : https://blog.logrocket.com/whats-new-nuxt-3/ [consulté le 18 juin 2023].

Improving Security of Nuxt 3, 2022*DEV Community* [en ligne]. Disponible à l'adresse : https://dev.to/jacobandrewsky/improving-security-of-nuxt-3-14ek [consulté le 20 juin 2023].

Intergiciel - Glossaire MDN : définitions des termes du Web | MDN, 2023 [en ligne]. Disponible à l'adresse : https://developer.mozilla.org/fr/docs/Glossary/Middleware [consulté le 21 juin 2023].

:icarus.gk sur Twitter, 2023 *Twitter* [en ligne]. Disponible à l'adresse : https://twitter.com/icarusgkx/status/1669091696548749319 [consulté le 26 juin 2023].

TypeScript, 2023 *Wikipédia* [en ligne]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=TypeScript&oldid=204321221 [consulté le 27 juin 2023]. Page Version ID: 204321221

JavaScript With Syntax For Types., [sans date] [en ligne]. Disponible à l'adresse : https://www.typescriptlang.org/ [consulté le 1e juillet 2023].

Qu'est-ce que TypeScript ? Un guide complet, 2023*Kinsta*® [en ligne]. Disponible à l'adresse : https://kinsta.com/fr/base-de-connaissances/guide-complet-typescript/ [consulté le 4 juillet 2023].

ECMAScript, 2023*Wikipédia* [en ligne]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=ECMAScript&oldid=205583144 [consulté le 30 avril 2023]. Page Version ID: 205583144

Inférence de types, 2023*Wikipédia* [en ligne]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=Inf%C3%A9rence_de_types&oldid=201377077 [consulté le 5 juillet 2023]. Page Version ID: 201377077

Intergiciel - Glossaire MDN : définitions des termes du Web | MDN, 2023 [en ligne]. Disponible à l'adresse : https://developer.mozilla.org/fr/docs/Glossary/Middleware [consulté le 5 juillet 2023].

Programmation impérative, 2023*Wikipédia* [en ligne]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=Programmation_imp%C3%A9rative&oldid=205566290 [consulté le 5 juillet 2023]. Page Version ID: 205566290

User Inyerface - A worst-practice UI experiment, [sans date] [en ligne]. Disponible à l'adresse : https://userinyerface.com/ [consulté le 6 juillet 2023].

JavaScript | MDN, 2023 [en ligne]. Disponible à l'adresse : https://developer.mozilla.org/fr/docs/Web/JavaScript [consulté le 6 juillet 2023].

JavaScript, 2023 *Wikipédia* [en ligne]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=JavaScript&oldid=206339758 [consulté le 9 juillet 2023]. Page Version ID: 206339758

Nitro - Create web servers that run anywhere., [sans date] [en ligne]. Disponible à l'adresse : https://nitro.unjs.io/ [consulté le 10 juillet 2023].

UI Kits for Developers and Web Agencies - cssninja.io, [sans date] [en ligne]. Disponible à l'adresse : https://cssninja.io/ [consulté le 12 juillet 2023].

Inertia.js - The Modern Monolith, [sans date] [en ligne]. Disponible à l'adresse : https://inertiajs.com/ [consulté le 30 juillet 2023].

Programming languages used in most popular websites, 2023*Wikipedia* [en ligne]. Disponible à l'adresse : https://en.wikipedia.org/w/index.php?title=Programming_languages_used_in_most_popular_websites&oldid=1162861465 [consulté le 14 juillet 2023]. Page Version ID: 1162861465

Nuxt sur Twitter, 4 novembre 2022 *Twitter* [en ligne]. Disponible à l'adresse : https://twitter.com/nuxt_js/status/1588535895883272192 [consulté le 16 juillet 2023].

Tweet sur Twitteer, 16 novembre 2022 *Twitter* [en ligne]. Disponible à l'adresse : https://twitter.com/nuxt_js/status/1592904778337906689 [consulté le 18 juillet 2023].

Showcase · Nuxt, [sans date] *Nuxt* [en ligne]. Disponible à l'adresse : https://nuxt.com/showcase?category=Featured [consulté le 24 juillet 2023].

NANDWANA, Vedant, 2023. *vedantnn71/nuxt-next-perf-test* [logiciel] [en ligne]. 13 juillet 2023. Disponible à l'adresse : https://github.com/vedantnn71/nuxt-next-perf-test [consulté le 25 juillet 2023].

ROGER!, 2023. *fw-bench* [logiciel] [en ligne]. 7 juillet 2023. Disponible à l'adresse : https://github.com/icarusgk/fw-bench [consulté le 26 juillet 2023].
HATOO, 2023. *oha (おはよう)* [logiciel] [en ligne]. Disponible à l'adresse : https://github.com/hatoo/oha [consulté le 26 juillet 2023].

XenForo - Compelling community forum platform, [sans date] [en ligne]. Disponible à l'adresse : https://xenforo.com/ [consulté le 26 juillet 2023].

<NuxtLink> · Nuxt Components, [sans date] [en ligne]. Disponible à l'adresse : https://nuxt.com/docs/api/components/nuxt-link [consulté le 30 juillet 2023].

🔐 *nuxt-auth* [logiciel] [en ligne]. 30 juillet 2023. sidebase.. Disponible à l'adresse : https://github.com/sidebase/nuxt-auth [consulté le 30 juillet 2023].

Auto-imports · Nuxt Concepts, [sans date] [en ligne]. Disponible à l'adresse : https://nuxt.com/docs/guide/concepts/auto-imports [consulté le 30 juillet 2023].

Data fetching · Get Started with Nuxt, [sans date] [en ligne]. Disponible à l'adresse : https://nuxt.com/docs/getting-started/data-fetching [consulté le 30 juillet 2023].

Deployment · Get Started with Nuxt, [sans date] [en ligne]. Disponible à l'adresse : https://nuxt.com/docs/getting-started/deployment [consulté le 30 juillet 2023].

:icarus.gk ⇏ ▾ sur Twitter, 2023*Twitter* [en ligne]. Disponible à l'adresse : https://twitter.com/icarusgkx/status/1669091696548749319 [consulté le 30 juillet 2023].

Installation · Get Started with Nuxt, [sans date] [en ligne]. Disponible à l'adresse : https://nuxt.com/docs/getting-started/installation [consulté le 30 juillet 2023].

Modules · Nuxt, [sans date] [en ligne]. Disponible à l'adresse : https://nuxt.com/modules [consulté le 30 juillet 2023].

Nitro - Create web servers that run anywhere., [sans date] [en ligne]. Disponible à l'adresse : https://nitro.unjs.io/ [consulté le 30 juillet 2023].

Rendering Modes · Nuxt Concepts, [sans date] [en ligne]. Disponible à l'adresse : https://nuxt.com/docs/guide/concepts/rendering [consulté le 30 juillet 2023].

Routing · Get Started with Nuxt, [sans date] [en ligne]. Disponible à l'adresse : https://nuxt.com/docs/getting-started/routing [consulté le 30 juillet 2023].

Security Module · Nuxt, [sans date] [en ligne]. Disponible à l'adresse : https://nuxt.com/modules/security [consulté le 30 juillet 2023].

Server Engine · Nuxt Concepts, [sans date] [en ligne]. Disponible à l'adresse : https://nuxt.com/docs/guide/concepts/server-engine#server-engine [consulté le 30 juillet 2023].

VeeValidate: Painless Vue.js forms, [sans date] [en ligne]. Disponible à l'adresse : https://vee-validate.logaretm.com/ [consulté le 30 juillet 2023].