MASTER EN SCIENCES INFORMATIQUES

Comparaison quantitative et qualitative de frameworks de développement web

Ziani, Ali

Award date:
2021

Awarding institution:
Universite de Namur

UNIVERSITÉ DE NAMUR

FACULTÉ D'INFORMATIQUE

Université de Namur
Faculté d'informatique
Année académique 2020–2021

Comparaison quantitative et qualitative de frameworks de développement web

Ali Ziani

Promoteur : _____ (Signature pour approbation du dépôt - REE art. 40)
Elio Tuci

Co-promoteur : Jean-Noël Colin

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

RUE GRANDGAGNAGE, 21 ● B-5000 NAMUR(BELGIUM)

Je voudrais remercier mes promoteurs monsieur Jean-Noël Colin et monsieur Elio Tuci pour leurs précieux conseils, leur patience et leur bienveillance. Je voudrais également remercier toutes les personnes qui m'ont soutenu lors de l'élaboration de ce mémoire de fin d'étude.

Résumé

Ces 10 dernières années, face à l'afflux de nouveaux frameworks Javascript, la sélection du meilleur framework est devenu un enjeu majeur dans le développement d'applications web côté frontend. Après une étude poussée de la littérature scientifique où nous avons questionné quelles étaient les méthodes utilisées, les frameworks concernés et les critères exploités, nous en avons conclus qu'aucune étude ou méthode étudiée ne permettait de sélectionner le bon framework de manière fiable. Nous avons alors proposé d'appliquer une méthode spécialement dédiée à la résolution de ce genre de problème, appelée AHP (analyse hiérarchique de procédés), elle permet de sélectionner un élément dans un groupe sur base de différents critères. Cette expérience nous a permis de construire un arbre de décision clair visuellement qui permet d'identifier les meilleurs et les pires frameworks par critère et pour tous les critères. Notre étude se démarque par notre transparence et nos explications détaillée de chaque étape de la réalisation de notre arbre de décision, chaque choix est argumenté et dûment justifié. Toutes divergences trouvées dans la littérature scientifique sont en faveur de notre étude. Néanmoins la méthode AHP n'est pas exempte de défauts, elle manque de flexibilité et certaines étapes si elles ne sont pas rigoureusement argumentées peuvent être subjectives, pour pallier à ces problèmes, une piste d'amélioration est proposée en conclusion.

Mot-clés— AHP, React, Vue, Svelte, Angular, Preact, effort apprentissage, effort développement, stabilité, attractivité, performance

Table des matières

Chapitre 1

Introduction

Vers la fin des années 1990, il était clair que le web aurait un impact phénoménal sur le monde [55], le nombre de sites web n'ayant fait qu'augmenter, en 2020 on dénombre déjà plus d'un milliard de sites web existant et accessible [55].
Si au départ l'objectif était de développer un site web pour pouvoir avoir un endroit facilement accessible et centralisant des informations intéressantes pour les entreprises privées/public et les États [2], la demande a vite évolué afin de proposer des applications permettant de proposer des services plus complexes et intéressant pour les utilisateurs finaux pour lequel une révision technologique a été rendue nécessaire [55, 56, 30]. Le web est basé sur un ensemble de technologie qui évolue très rapidement, de nouvelles architectures et librairies voient le jour à un rythme soutenu et de plus en plus fréquent [55, 28]. Le choix du bon framework a potentiellement plusieurs impacts : sécurité, performance, qualité de code, ... Il est donc crucial de faire le bon choix [19].

Le développement d'applications web combinant plusieurs outils et technologies et étant basés sur des alternatives de l'architecture client/serveur [29, 11], plusieurs questions se posent : faut il centraliser la logique métier de l'application sur le serveur, sur le client ou les deux ? Quel langage serveur choisir (ex : Java, C#, C, Php, Javascript,...) ? Quel langage client choisir (ex : Javascript, webassembly, ...) ? Faut-il basé l'architecture de la partie cliente de notre application sur un framework (Angularjs, Reactjs, Vuejs, ...) ? Quelles types de tests faut il implémenter (ex : unit test, webtest, integration test, ...) [21] ?
Aurons-nous besoin de librairies d'aide au développement (ex : Webpack, Gulp, Grunt, ...) ? Quelle librairie de présentation allons nous implémenter (ex : SASS, LESS, CSS, ...) ? Toutes ces questions n'ont pas le même poids critique mais doivent être posées à l'initialisation du projet.

Lors de cette étude nous concentrerons notre recherche sur le choix du bon framework Javascript à implémenter dans le cadre du développement d'une application web côté frontend pour différentes raisons : d'abord plusieurs personnes alertent que cette problématique représente un enjeu majeur dans le cadre du développement web [24, 56, 11, 29, 28], ensuite entre 71.5% [55] et 88.2% [28] des sites web l'utilisent, ce qui en fait le langage le plus utilisé dans le monde.

Ce travail a pour objectif de répondre à ces questions :

— QR1 : Existe t'il une méthode efficace de comparaison et sélection d'un framework Javascript dans le cadre du développement d'une application

web ?
- QR2 : Quels sont les critères retenus pour comparer des framework Javascript ?
- QR3 : Est-ce que tous les frameworks Javascript sont pris en compte ?

La littérature scientifique comporte de nombreuses comparaisons de frameworks ou de librairies mais nous allons découvrir que celles-ci ne portent que sur un nombre limité de choix à la fois. Aucun standard ou référence proposant une méthodologie complète d'évaluation et de sélection de framework ou librairie éprouvée dédiée au développement d'applications web n'a été trouvé.

Afin de répondre à ces questions, nous commencerons par consacrer un 2ème chapitre afin de présenter l'origine de cette technologie, ensuite nous mettrons en évidence qu'il existe plusieurs définitions reconnues de ce qu'est une application web, nous listerons les plus importantes et nous nous positionnerons par rapport à cette problématique. Nous présenterons également quelques moments et concepts que nous jugeons les plus importantes à comprendre concernant le développement de client dynamique et le langage Javascript.
Dans le 3ème chapitre, nous présenterons notre méthodologie de recherche de la littérature, nous y présenterons les étapes que nous avons suivi ainsi que le nombre final d'articles retenu.
Dans le 4ème chapitre, nous présenterons de manière critique les différentes comparaisons existant dans la littérature ce qui nous permettra de répondre à toutes nos questions de recherche.
Dans le chapitre 5, nous identifierons clairement notre objectif, nous expliquerons la méthodologie choisie (AHP) et l'appliquerons, ensuite nous discuterons des résultats obtenus. Le dernier chapitre sera dédié à la conclusion de notre recherche.

Chapitre 2

Fondamentaux

2.1 Origine du web

Le web est un système de partage d'information développé par Tim Berners-Lee en 1989 au Cern et présenté au public en 1992. Il est parfois cité par la contraction "w3" [2]. Dans leur article rédigé avec R. Cailliau and J.-F. Groff, ils y expliquent les objectifs à l'origine de cette invention : [2]

— Centraliser tout type d'information dans une interface intuitive pour permettre à n'importe qui d'y accéder
— Augmenter la quantité et la qualité de l'information en ligne en facilitant l'ajout d'informations

Trois problèmes avaient été identifiés :

— Le problème de portabilité entre plateformes de l'époque
— Le format des données
— Le protocole d'accès

Le web est conçu selon une architecture client/serveur, le client permettant de visualiser des documents accessibles via une adresse unique, appelée URL, ces documents peuvent se référencer les uns les autres via des liens hypertextes. Ces premières pages web sont hébergées par le serveur qui centralise les algorithmes de recherche et de manipulations de données [3, 2, 33]. Plus bas vous pouvez voir dans la figure 2.1 l'idée qu'ils se faisaient de l'infrastructure nécessaire à la mise en place d'une telle solution.

Le problème de portabilité a été résolu par la création des premiers navigateurs web : Erwise, Viola, ... Le choix du format de données s'est porté sur le langage HTML basé lui-même sur SGML pour plusieurs raisons : il était communément accepté comme format d'encodage, il était beaucoup utilisé dans la rédaction de documentation et pour finir il est assez agile et flexible pour permettre d'encoder différents types de données et d'encapsuler des formats non reconnus par SGML. Le protocole HTTP a été créé pour permettre de récupérer ces documents [2].

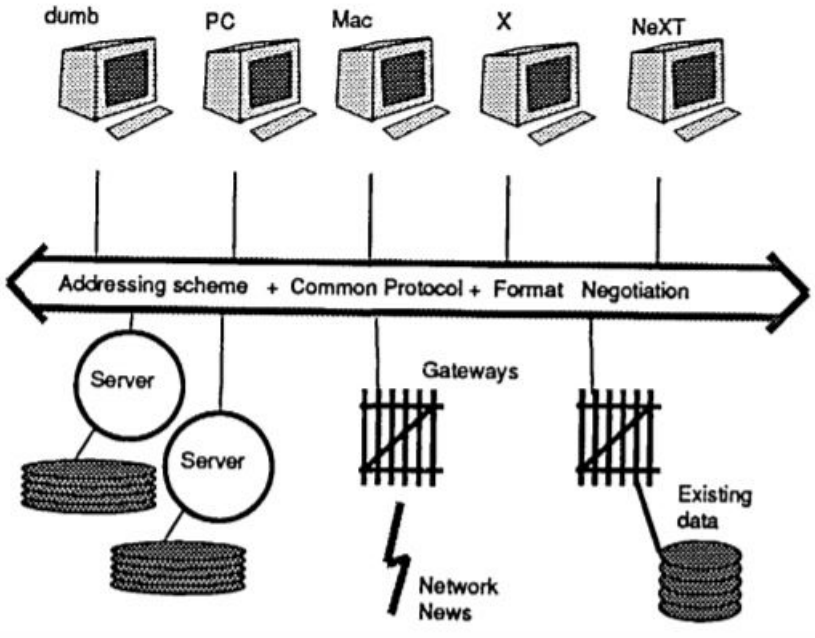

FIGURE 2.1 – Architecture originale du web [2]

2.2 De la page web à l'application web

Ensemble de pages web vs site web

À l'époque seules les institutions académiques (ex : universités) et les entreprises connectées à Internet disposaient de l'infrastructure nécessaire à l'hébergement de pages web, malgré la rareté des ordinateurs personnels, le web a connu une croissance exponentielle due à la facilité de création de ces pages, un simple éditeur de texte était amplement suffisant pour écrire du contenu HTML. Shklar Leon et Rosen Richard avancent dans leur livre que très vite une distinction s'est faite entre les pages web et les sites web. D'après eux, un site web est plus qu'un ensemble de pages web se référençant les unes les autres [33]. Ils avancent plusieurs points distinctifs :

- La consistance du contenu : les pages doivent partager la même thématique
- L'esthétique : le layout, la typographie et le design doivent être cohérente sur toutes les pages
- L'agencement des pages : l'agencement du contenu doit être organisé de sorte à être intuitive et logique

Début du web dynamique

Malgré l'exposition dès le début du potentiel dynamique de ces applications par Tim Berners-Lee qui avait démontré que c'était possible en créant une page web listant des numéros de téléphones présents dans une base de données, il a fallu plusieurs années avant que ce type de développement ne se généralise, en

effet Shklar Leon et Rosen Richard rappellent qu'au début du web la grande majorité des sites web consistaient en de simples sites présentant du contenu HTML statique [33].

Dans leur livre, Shklar Leon et Rosen Richard rappellent néanmoins que les plus anciennes spécifications de logiciels dédiés au développement de pages web dynamique étaient disponibles depuis 1993 [33].

Site web vs application web

La définition d'une application web n'a pas toujours été claire et unanime. En 1999, Conallen définissait une application web comme

> "un système web (serveur, réseau, http, navigateur) où l'entrée de l'utilisateur (navigation and donnée encodée) affecte l'état du business." [4]

La même année, Gellersen et Gaedke ont défini une application web comme

> "étant un logiciel applicatif qui dépend du web pour sa bonne exécution." [8]

En 2002, Taylor et Fielding ont fourni une définition plus détaillée

> "une application web constitue un réseau de pages web formant une machine à état virtuel, permettant aux utilisateurs de progresser dans leur visite en cliquant sur un lien ou soumettant un formulaire, chaque action étant liée à une transition à l'état suivant de l'application qui met à jour la représentation de ce nouvel État à l'utilisateur final." [6]

Celle qui nous semble être la plus complète, est celle utilisée par Shklar Leon et Rosen Richard en 2003 qui définissent

> "une application web comme une application client/serveur qui utilise le navigateur web comme programme client et fournit des services interactifs par l'intermédiaire d'un serveur accessible à partir d'internet. Un site web fournit simplement du contenu statique alors qu'une application web présente en plus du contenu dynamique basé sur les interactions de l'utilisateur, permet de suivre le comportement de l'utilisateur et prends en considération les aspects de sécurité (ex : vérification des requêtes, limitations des droits, etc...) [33]."

2.3 Client dynamique

Il a été très tôt rendu nécessaire de permettre de dynamiser une page web directement sur le navigateur pour différentes raisons telles que : animations, optimisation de la bande passante, ... [55]

En 20 ans plusieurs solutions techniques ont été proposées et utilisées (ex : SilverLight, Flash, Java Applet, ...), mais seul le langage Javascript a subsisté avec ses différentes librairies et parser (ex : GWT, CoffeeScript, Typescript, WebAssembly, ..) [55]. Les autres solutions sont soit dépréciée (ex : Flash), supprimée (ex : SilverLight) ou vivement déconseillée (ex : Java Applet) [55]. Comme le rappelle Allen Wirfs-Brock et Holger M. Kienle dans leur article, le

langage Javascript a été créé par l'entreprise Netscape et a été conçu à l'origine pour être simple, facile à utiliser, facile à apprendre et facile à intégrer dans une page web. Le code est interprété par le navigateur au chargement de la page afin de personnaliser dynamiquement la présentation et répondre aux interactions de l'utilisateur [55, 21].

Les concepteurs du langage Javascript étaient conscients dès le début du besoin de spécification afin d'assurer l'interopérabilité des pages web entre les différents navigateurs web.
Pour cela plusieurs entités ont été consultées telles que W3C ou IETF mais comme la standardisation des langages de programmation n'était pas leur spécialité il a fallu s'orienter vers un autre acteur.
Ils avaient également peur que Microsoft obtienne un monopole sur les technologies du web et tente de remplacer le langage Javascript par un langage leur appartenant déjà, c'est la raison pour laquelle ils se sont orienté vers l'instance de standardisation ECMA [55].

Au fur et à mesure que le contexte du web s'est complexifié, d'autres navigateurs ont été créé dont voici une liste non exhaustive : Opera (1994), Internet Explorer (1995), Mozilla Firefox (2004), Apple Safari (2003), Google Chrome (2008), ... [55]

D'après Allen Wirfs-Brock et Holger M. Kienle, le problème d'interopérabilité du langage Javascript entre navigateurs est principalement dû à 2 problèmes, d'abord les premières versions des spécifications du langage étaient incomplètes et imprécises ce qui a mené les concepteurs des différents navigateurs à implémenter leur moteur sur base de leur interprétation mais également de fournir une API plus fournie ou moins fournie en fonction des cas, ensuite les utilisateurs à l'époque ne mettaient à jour leur navigateur que rarement ou lorsqu'ils changeaient d'ordinateur, ils ne disposaient donc pas toujours de la version du navigateur la plus à jour [55].

À l'époque, Allen Wirfs-Brock et Holger M. Kienle rappellent qu'il existait principalement 2 manières de gérer la différence de version du langage dans les navigateurs au niveau applicatif, la première était de créer une version de leur application web par navigateur incompatible, ils identifiaient la provenance et renvoyaient la version adéquate, cette manière a apporté beaucoup de difficulté au niveau de la maintenance. L'autre solution était de garder une seule version de l'application mais de prendre en compte tous les cas de figure par navigateur dans le source code, ce qui devenait également très vite ingérable [55].

Cette problématique est selon Allen Wirfs-Brock et Holger M. Kienle l'une des raisons principale pour laquelle on a eu une émergence des frameworks et librairies [55].
Ils citent notamment : Prototype, Mootools, Doko, JQuery, ... Ces frameworks assuraient l'interopérabilité et fournissaient une API fournie permettant d'accélérer le développement d'une application web[55].

Chapitre 3

Méthodologie de recherche

Notre recherche porte sur les méthodes d'évaluation et de sélection de frameworks Javascript dans le développement d'applications web côté frontend. Nous avons en particulier recherché les articles établissant une revue de ces méthodes sur les moteurs de recherche suivants : IEEE Xplore ; ACM ; Science Direct. ResearchGate a uniquement été utilisé dans la phase de snowballing. Nous avons utilisé le programme Zotero comme gestionnaire d'articles [49].

La construction de notre requête est basée sur nos 3 questions de recherches. Nous avons défini comme critère d'exclusion les articles dupliqués et tous les articles qui n'étaient pas rédigés en langue anglaise. En plus de ces 2 critères nous avons filtré nos articles en 3 phases :

— Lecture du titre
— Lecture du résumé (abstract)
— Lecture complète de l'article

Les moteurs de recherche utilisés, la chaîne de recherche déduite ainsi que les résultats obtenus sont indiqués dans la table 3.1. La colonne « F0 » y indique le nombre de résultats retournés par le moteur correspondant, la colonne « F1 » indique le nombre d'articles restant après lecture des titres, la colonne « F2 » indique le nombre d'articles restant après la lecture des résumés et la colonne « F3 » y indique le nombre de publications restantes après une lecture approfondie des articles.

Après analyse des résultats retournés par les différents moteurs de recherche, nous avons retenu 8 articles. Après lecture approfondie de ces articles, nous avons rajouté 6 autres articles par snowballing.

— Par référence : 1
— Par citation : 5

Note : Nous sommes conscients que l'utilisation du "wildcard" dans nos requêtes aurait permis de récupérer potentiellement plus de cas mais le moteur de recherche Science Direct ne l'acceptant pas et dans un souci de conformité nous avons pris la décision de ne pas l'utiliser également pour les autres moteurs de recherche.

Moteur	**Requête**	**F0**	**F1**	**F2**	**F3**
IEEE Xplore	("web development" OR "web application" OR "web project") AND (jsf OR "javascript framework" OR angularjs OR react.js OR vue.js) AND (comparing OR compare OR comparating OR comparison OR select OR selection OR evaluate OR evaluation)	13	9	2	2
ACM	("web development" OR "web application" OR "web project") AND (jsf OR "javascript framework" OR angularjs OR react.js OR vue.js) AND (comparing OR compare OR comparating OR comparison OR select OR selection OR evaluate OR evaluation)	363	51	11	6
Science direct	("web development" OR "web application" OR "web project") AND (jsf OR "javascript framework" OR angularjs OR react.js OR vue.js) AND (comparing OR compare OR comparating OR comparison OR select OR selection OR evaluate OR evaluation)	76	14	3	0

TABLE 3.1 – Requêtes de collecte de publications par moteur de recherche.

Chapitre 4

Comparaisons existantes

4.1 Résultats brut chronologiques

Dans leur article en 2012, Andreas B. Gizas, Sotiris P. Christodoulou et Theodore S. Papatheodorou [10] affirment qu'il faut prendre en compte 3 aspects quand on compare des frameworks Javascript : la qualité, la performance et les tests de validation fournis par le framework.
Ils mesurent la qualité en se basant sur des unités de mesure classique telle que : le nombre de lignes de code, la complexité (ex : complexité cyclomatique) et la maintenabilité [10].
Ils comparent quantitativement 6 frameworks : ExtJS, Dojo, jQuery, MooTools, Prototype et YUI en comparant les fonctionnalités de base telle que : [10]

- La possibilité de manipuler le DOM
- Les sélecteurs
- Les requêtes ajax
- Création et utilisation de formulaire
- Gestion d'évènements
- Support de la compatibilité entre navigateurs

Les outils utilisés pour les tests de qualité sont : JSmeter, CLOC et Understand. Les tests de validation ont été menées avec le logiciel Yasca en combinaison avec Javascript Lint [10]. Ils ont utilisé plusieurs systèmes d'exploitation et navigateurs pour leurs tests de performance qu'ils ont analysé avec le framework de test SlickSpeed Selectors [10]. Leur objectif n'était pas de cibler le meilleur framework mais de mettre en évidence certaines améliorations à appliquer ou défauts à corriger. Les résultats obtenus sont discutables (voir figure 4.1), la plupart de ces frameworks et des systèmes d'exploitation et navigateurs en tant que tels ou leurs versions ne sont plus maintenus (ex : Windows Xp, Mootools, ...), il faudrait refaire la même étude avec des frameworks et des systèmes d'exploitation et des navigateurs à jour [10].
Il est également à noter que cet article est cité par plusieurs articles que nous allons présenter plus bas [19, 28, 22, 24].

Dans leur article en 2012, Sanjay Misra et Ferid Cafer [23] proposent une nouvelle unité de mesure appelée JCCM permettant d'évaluer le niveau de qualité d'un projet Javascript (quelle que soit son implémentation).
La formule consiste en la somme de 5 valeurs :

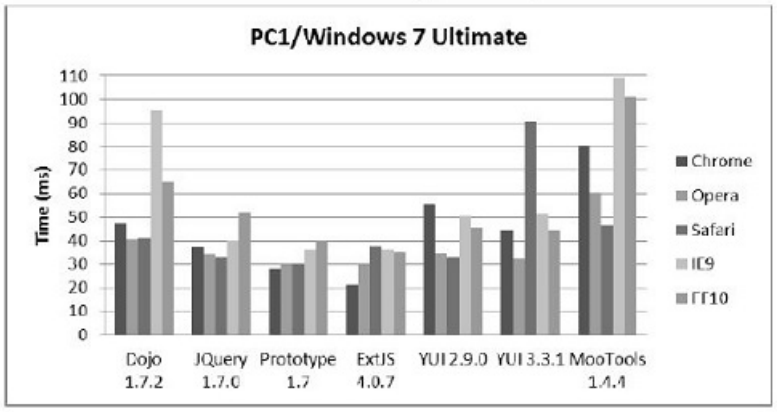

FIGURE 4.1 – Performance des frameworks par navigateur [10]

- Le nombre total de lignes de code
- Le nombre de variables nommées arbitrairement (dont la lecture ne permet pas de comprendre l'objectif de la variable)
- Le nombre de variables nommées de manière significative
- La somme des poids (basés sur une table de valeur) de toutes les structures de contrôles (séquence, condition, boucle, exception, appel de fonctions)
- Le nombre d'opérateurs

Les auteurs ont ensuite appliqué cette formule sur 30 fichiers Javascript différents [23].
Ils ont comparé les résultats avec d'autres unités de mesure telle que le nombre cyclomatique , le nombre logique de lignes de code et l'unité de mesure Halstead.
Les auteurs affirment prendre en compte tous les facteurs impliquant une hausse de la complexité dans un fichier Javascript et en évaluant les résultats des comparaisons avec les autres unités de mesure, ils affirment que leur unité de mesure est la meilleure [23].
Ils ne font pas mention et nous n'avons pas trouvé d'autres articles reprenant cette unité de mesure pour comparer des simples fichiers ou des frameworks Javascript.

Dans leur article en 2013, Daniel Graziotin, Pekka Abrahamsson [19] proposent une nouvelle méthode de comparaison de frameworks Javascript mêlant quantitatif et qualitatif, se basant sur la première étude [10].
Leur proposition consiste en l'ajout de 3 critères identifiés suite à une discussion avec 4 développeurs frontend qui selon eux sont les principaux qui mènent les développeurs vers un choix de framework

- La documentation
- La taille et le niveau de participation de la communauté
- Les fonctionnalités du framework et certaines informations complémentaires (maturité, cycle de mise à jour)

Ils insistent également sur le fait qu'il faille implémenter une solution concrète comme par exemple une TodoList par framework afin de pouvoir comparer concrètement les différentes solutions. Dans la figure (voir figure 4.2), vous pou-

FIGURE 4.2 – Comparaison de frameworks [19]

vez voir leur proposition [19].
Le principal problème que nous avons avec cet article est que nous disposons pas d'informations concrète sur le questionnaire utilisé par les 4 développeurs, nous n'avons aucune information sur leur expérience, en dehors du fait que ces critères semblent être purement subjectifs nous pensons que le nombre de développeurs contactés est trop petit (en comparaison avec [31] par exemple) [19].
Nous n'avons pas trouvé d'exemple concret de comparaison de frameworks qui utilise cette méthode.

Dans leur article en 2016, Miguel Ramos, Marco Tulio Valente, Ricardo Terra et Gustavo Santos [31] se concentrent sur un framework en particulier : AngularJs (version 1), il est important de noter que la version du framework étudié n'est plus maintenue, Angular est à l'heure actuelle à sa version 10.
Ils ont créé une enquête auprès de 460 développeurs afin de leur demander quelles sont les fonctionnalités qu'ils apprécient le plus et celles qu'ils apprécient le moins [31]. Cette étude permet de mettre en avant que les fonctionnalités les plus intéressantes pour les développeurs sont entre autres la dépendance d'injection, la possibilité de personnaliser ses composants, double liaison, facilitation d'écriture de tests automatisés, facilitation de liaisons d'évènements avec le DOM, pouvoir disposer de POJO [31]. Elle permet également d'identifier les nouveaux problèmes que ça génère : dégradation de la performance, complexité d'implémenter certains concepts (ex : directive), les erreurs sont mal gérées, complexité supplémentaire pour la compréhension, ... [31]
Cet article n'est pas utile en tant que telle pour notre objectif, mais permet de mettre en avant les fonctionnalités attendues par les développeurs à l'époque pour ce genre de framework. Nous sommes conscients que la liste n'est pas exclusive, sachant qu'elle a été réalisée sur base d'un seul framework mais elle nous permet d'avancer concernant notre question de recherche QR2.

Dans leur article en 2016, Jaime Raigoza et Rushi Thakkar [29] comparent quantitativement les frameworks JQuery et SAPUI5 sur leurs performances.
Leur objectif était de justifier l'utilisation de SAPUI5 si l'alternative était JQuery sachant que SAPUI5 est construit sur base de JQuery [29] .
Pour comparer ces frameworks, ils ont créé deux applications TodoList [29] et utiliser un outil développé par Microsoft pour exécuter les tests de performances [48].
Les résultats de leurs tests permettent de mettre en évidence que JQuery met plus de temps à charger des données (voir figure 4.3) et SAPUI5 met plus de temps à charger du contenu graphique (voir figure 4.4).
Il n'y a donc pas de réponse catégorique.

Dans leur article en 2017, Miguel Ramos, Marco Tulio Valente et Ricardo

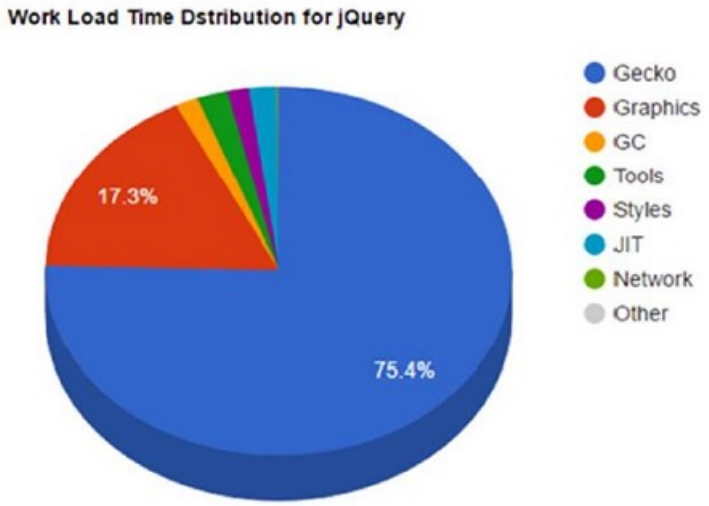

FIGURE 4.3 – JQuery performance distribution [29]

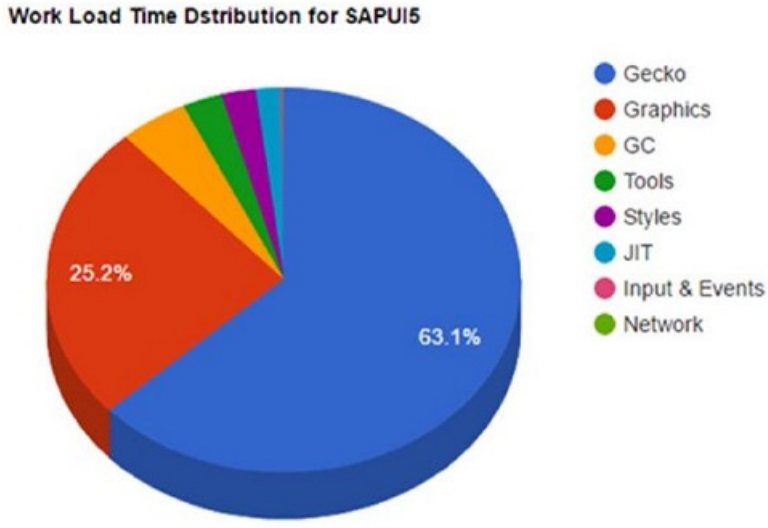

FIGURE 4.4 – SAPUI5 performance distribution [29]

Terra [30] ont mené une nouvelle enquête sur une population de 95 développeurs pour identifier l'origine d'après eux des problèmes de performance des applications web utilisant AngularJs (version 1). Tous les points identifiés se rassemblent sur une même cause, les développeurs utilisent des fonctionnalités qu'ils ne maîtrisent pas ce qui les amènent parfois à utiliser des fonctionnalités appliquant des logiques complexes pour résoudre des problèmes très simple (ex : pas besoin d'utiliser la double liaison si le contenu de la variable qu'on affiche ne sera plus jamais mise à jour par la suite) [30]. Cet article nous aide indirectement pour répondre à la question de recherche QR2.

Dans leur article en 2017, Amantia Pano, Daniel Graziotin et Pekka Abrahamsson [28] étudient quels sont les facteurs qui mènent les développeurs à choisir un framework Javascript plutôt qu'un autre sans donner d'exemple concret. Pour cela, ils ont fait un état des connaissances assez large en allant chercher plus loin que la littérature scientifique (ex : blogs, tutoriels, ...) ce qui les a permit de créer un questionnaire qu'ils ont utilisé pour mener une enquête qualitative auprès de 18 professionnels. [28]
Cette enquête leur a permis de proposer un modèle de comparaison regroupant les critères les plus couramment utilisés qu'ils regroupent ensuite par catégories (voir figure 4.5) [28].
Ces catégories sont : l'attente en terme de performance, la charge d'effort, l'influence sociale, l'accessibilité et finalement le coût [28].

Dans leur article en 2019, Michael Xieyang Liu et Brad A. Meyers [22] proposent une extension Google Chrome pour compiler les arguments menant à un choix technologique sous la forme d'un tableau où les colonnes représentent les critères, les lignes représentent les options et les cellules peuvent contenir autant d'arguments que le développeur désire mais ceux-ci doivent être catégorisés, il existe 3 types de catégories informatif (information neutre), positif (argument pouvant mener à la sélection de l'option), négatif (contre-argument).
Il est ensuite possible de partager le résultat et d'accéder à un historique présentant une ligne de vie de la composition des arguments (voir figure 4.6) [22].
Les auteurs avancent 2 arguments pour vendre leur outil [22] d'après eux la manière dont l'information est présentée facilite la compréhension de la prise de décision, ensuite l'utilisation de leur extension permet de gagner 45% du temps investi dans la collecte et la synthèse de l'information comparée à Google Doc.
Néanmoins l'extension présente 2 limitations : il n'est pas possible de prioriser les critères, il n'est pas possible de travailler collaborativement sur un même projet [22].
L'outil n'est pas encore utilisé dans le monde professionnel, le seul retour que nous avons trouvé est une vidéo de présentation Youtube [57]. Nous n'avons pas trouvé de références utilisant ou critiquant cette outil dans la littérature scientifique.

Dans leur article en 2019, YongKang Xing, JiaPeng Huang et YongYao Lai [56] comparent quantitativement les frameworks AngularJs, Angular 2, ReactJs et VueJs sur base de 4 critères : traitement de données (simple ou double liaison), volume et performance, la spécification minimum requise du langage Javascript, le support technique.
Il en ressort que [56] dans le cadre du développement d'un e-commerce il est fortement conseillé de choisir le framework Angular 2 alors que React et Vue sont plus adaptés pour des applications plus petites ou moyennes telles que : blog, communication, ... [56]

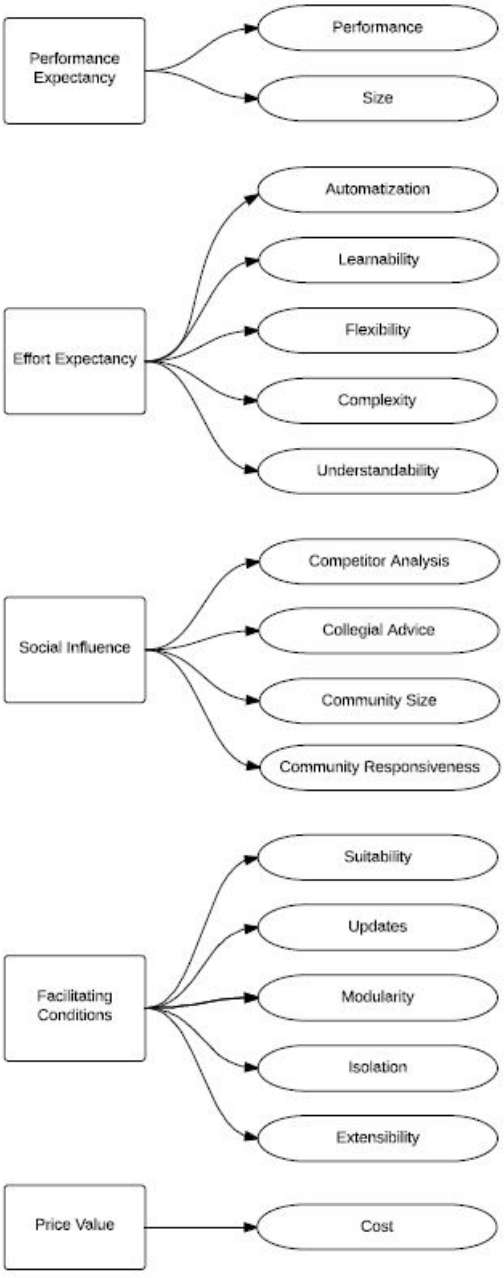

FIGURE 4.5 – Catégories de critères proposés [28]

Dans leur article en 2019, Ankush Ravindra Gochke et Mr. Prakash Kene [11] comparent 4 frameworks : Angular, React, Ember et Backbone.
Leur méthodologie de comparaison n'est pas claire, les critères utilisés sont subjectifs et ne se base sur aucune méthode ou standard connu [11].
Les auteurs arrivent à la conclusion que le meilleur framework à implémenter quel que soit le contexte est ReactJs [11]. Ce résultat catégorique rentre en contradiction avec les résultats obtenus dans l'article précédent et nous font douter du résultat de cette étude [56].

Dans leur article en 2019, Nozomi Nakajima, Shinsuke Matsumoto et Shinji Kusumoto [24] ont identifié 3 problèmes majeurs dans la comparaison de framework Javascript.

- Le nombre de frameworks existant est important et continue de grandir
- Ces frameworks sont destinés à fournir de base à différents types d'application exécutés sur des environnements différents qui peuvent nécessiter pour chacun des dispositions complémentaires (ex : polyfill)
- Les données et la documentation dont on dispose et les comparaisons déjà réalisées précédemment peuvent être dépassées voire même incorrectes au jour d'aujourd'hui.

Pour répondre à ces différentes problématiques, les auteurs ont développé un nouvel outil appelé "Jact" permettant de faciliter la comparaison de frameworks.
Il permet de faire de la comparaison de code source, d'exécuter des tests de performance basés sur des tâches simples comme la manipulation du DOM ou l'exécution de requête ajax. Pour évaluer l'utilisabilité de leur outil, ils ont fait 2 expérimentations.
D'abord les auteurs ont comparé les résultats de performance de manipulation du DOM de 2 frameworks : Jquery et VueJs et également le langage Javascript sans framework sur 2 navigateurs webs Firefox (voir figure 4.7) et Google Chrome (voir figure 4.8), il en résulte que ne pas utiliser de framework est le plus rapide (ce qui semble naturel, limité les intermédiaires ne peut qu'améliorer à notre sens la rapidité d'exécution) et que le framework VueJs est le plus lent.
Ensuite, ils ont mis au point une expérimentation réalisée par 13 personnes et leur ont soumis ensuite un questionnaire pour recevoir un feedback.
Ils ont obtenu des retours plutôt positifs, mais plusieurs pistes d'améliorations ont été évoqués, comme par exemple actuellement il n'est pas possible de comparer plus de 2 frameworks en même temps,... Le code source de l'outil n'est malheureusement pas rendu disponible, seul un lien direct vers une page de test est rendu public [24]. Nous n'avons pas trouvé de références utilisant ou critiquant cet outil dans la littérature scientifique.

4.2 Discussion

4.2.1 Méthodes de comparaison

Dans la littérature scientifique nous avons 3 méthodes de comparaisons

- Il existe des outils créés à l'occasion qui soit exécutent des batteries de tests qui permettent d'identifier le meilleur framework soit offre une manière intuitive de présenter les arguments menant à un choix, en dehors des articles eux-mêmes faisant l'éloge de leur solution, nous n'avons

pas trouvé d'autres articles promouvant leur qualité ou présentant un exemple d'utilisation [24, 22].
- Il existe une approche mathématique, des critères mesurables sont proposés et combinés dans un tableau, un graphe ou dans une formule sous la forme d'une addition et permettent d'obtenir un résultat parfois catégorique [10, 23, 19, 29].
- Sinon il existe une approche plus subjective qui se base sur l'avis de professionnels, d'informations statiques (qui peuvent changer dans le temps) et sur certains critères comme l'engouement du framework sur Github ou Stackoverflow, la taille de la communauté, ... [31, 30, 28, 56, 11]

L'implémentation d'une application web simple mais complète reste vivement conseillée afin de pouvoir expérimenter la différence de développement et d'exécution [19, 29, 24].

Pour répondre à la question de recherche QR1 : Existe t'il une méthode efficace de comparaison et sélection d'un framework Javascript dans le cadre du développement d'une application web ? Il semble qu'il n'existe pas de méthode standard de comparaison valide et intemporel.

4.2.2 Critères de comparaison

Pour répondre à la question de recherche QR2 : Quels sont les critères retenus pour comparer des framework Javascript ?
Dans la littérature scientifique nous trouvons plusieurs groupes de critères différents.
D'abord nous retrouvons les critères "qualitatif" mesurables :
- Mesure de la taille : comprends le nombre de lignes, le nombre d'instructions, le nombre de commentaires et le ratio du nombre de commentaires par rapport au nombre de lignes [10, 28]
- Mesure de la complexité : comprends la complexité cyclomatique de McCabe, le nombre de branches et la profondeur [10]
- Mesure de la maintenabilité : comprends la mesure d'Halstead et l'index de maintenabilité [10]
- Mesure de la performance : consiste à mesurer le temps que prend l'exécution de fonctionnalités spécifiques avec différents navigateurs sur différents systèmes d'exploitation [10, 29, 28, 24]
- Javascript Cognitive Complexity Measure (JCCM) : formule regroupant le nombre total de lignes de code, le nombre de variables nommées arbitrairement, le nombre de variables nommées de manière significative, la somme des poids de toutes les structures de contrôle et finalement le nombre d'opérateurs [23]

Ensuite les critères correspondant aux fonctionnalités attendues par les développeurs :
- Documentation correcte, intuitive et à jour : elle devrait contenir des exemples pratiques pour faciliter l'apprentissage [19, 28]
- Double liaison : implémentée par plusieurs frameworks (Angular, Vue, React, Knockout, ...), cette fonctionnalité permet une mise à jour automatique de la valeur d'une propriété si elle est modifiée dans le modèle ou dans l'interface utilisateur [31]
- Dépendance d'injection : permet de gérer les dépendances automatiquement, le gain est une réduction du couplage et une augmentation de la

testabilité.[31]
- Support pour la définition de composant personnalisable [31]
- Facilité d'écriture de tests automatisés [31]
- Facilité la liaison d'évènement directement dans le template HTML [10, 31]
- Pouvoir disposer de POJO pour la création de modèle (rien de superflu) [31]
- JSX : extension du Javascript pour permettre de manipuler du HTML avec du code Javascript [11]
- Manipulation du DOM [24]
- Requête ajax [24]
- Formulaire [24]
- Sélecteurs [10]

En troisième le résultat de l'utilisation d'outils externes exécutant des tests automatisés :

- Yasca : c'est un programme open source qui cherche des vulnérabilités liées à la sécurité mais évalue également la qualité du code, la performance et la conformité aux bonnes pratiques [10]
- Jact : permet de comparer le code source de 2 projets développés à partir de frameworks différent et d'exécuter des tests de performances sur chaque framework [24]

Finalement les critères statiques ou mesurable mais peu fiable car changeant, qui rassurent les développeurs :

- Participation de la communauté : mesurée par le nombre d'étoiles du projet s'il est hébergé sur github, ou par le nombre de requêtes ouvertes et traitées sur stackoverflow et la rapidité à laquelle une réponse à une question peut être reçue [19, 28]
- Maturité du framework [19]
- Fréquence des mises à jours [19, 28]
- Mesure du degré d'effort : rassemble l'automatisation, l'apprentissage, la flexibilité, la complexité et la compréhension [28]
- Prix : rassemble tous les coûts nécessaires à l'exploitation du framework donc licence, formation, certifications, ... [28]
- La version du langage minimum nécessaire pour utiliser le framework [56]
- Support technique : autre que les forums existants (ex : stackoverflow, developpez.com, ...) [56]

4.2.3 Frameworks comparés

Pour répondre tout de suite à la question de recherche QR3 : Est-ce que tous les frameworks Javascript sont pris en compte ?
Aucun article trouvé ne permet de comparer directement tous les frameworks existants, les frameworks impliqués dans la comparaison sont soit issu d'un objectif personnel et spécifique [29], soit d'un effet de mode [31, 30, 11].
Il est également intéressant de noter que des frameworks analysés il y a 8 ans sont déjà complètement dépassé [10] et que la comparaison du trio de framework Angular, ReactJs et VueJs reviennent le plus souvent dans les articles récents [31, 30, 56, 11].

4.3 Constat

Nous avons consacré notre étude sur la comparaison qualitative et quantitative des frameworks Javascript dans le cadre du développement web côté frontend qui nous a mené à identifier plusieurs questions de recherche :

- QR1 : Existe t'il une méthode efficace de comparaison et sélection d'un framework Javascript dans le cadre du développement d'une application web ?
- QR2 : Quels sont les critères retenus pour comparer des framework Javascript ?
- QR3 : Est-ce que tous les frameworks Javascript sont pris en compte ?

Pour la question de recherche QR1, nous avons trouvé plusieurs comparaisons dans la littérature scientifique, mais il semble qu'il n'existe pas de standard ou de comparaison unanimement reconnue, correct et intemporel.
Pour la question de recherche QR2, notre recherche nous a permis d'identifier 4 groupes de critères :

- Les critères mesurables
- Les fonctionnalités désirées/attendues
- Le résultat de l'exécution de logiciels externes
- Les critères statiques ou mesurables mais peu fiables car changeant

Finalement pour la question de recherche QR3, nous n'avons pas trouvé de solution intemporelle, de nouveaux frameworks sont créé tous les ans, les frameworks existants sont améliorés et parfois refactoré sans offrir de rétrocompatibilité (ex : Angular v1 vs Angular v2) et finalement il peut arriver que des frameworks n'assurent plus de support.
Ce constat implique quelle que soit la solution mise en place d'un coût de maintenabilité afin de gérer les nouvelles adaptations (création, modification, nouvelle version, suppression).

Ces réponses à nos questions de recherche, nous amènent à proposer une manière différente de comparer les frameworks Javascript et de les représenter. Notre idée serait d'utiliser les arbres de décision.

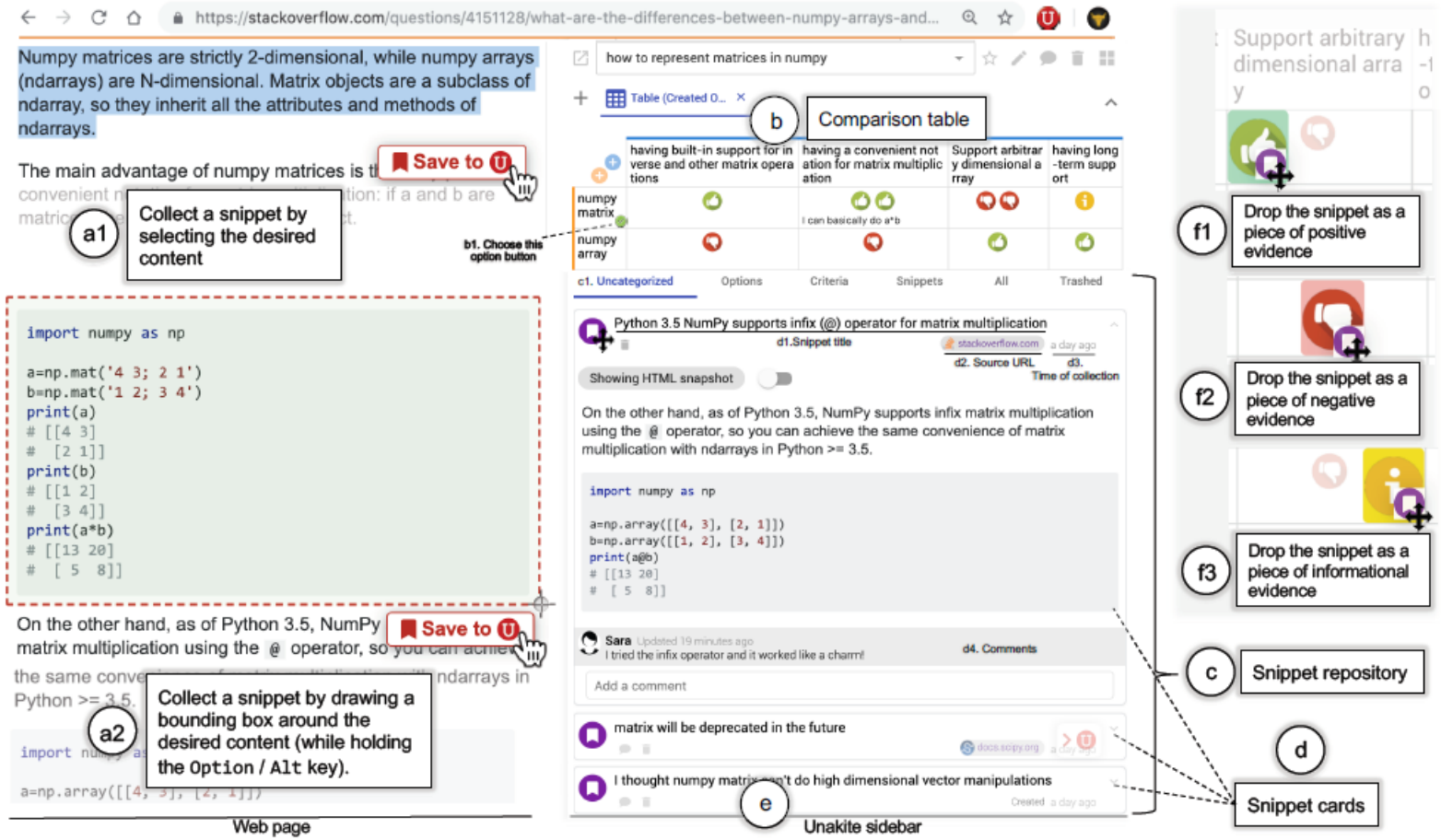

FIGURE 4.6 – Présentation de l'outil [22]

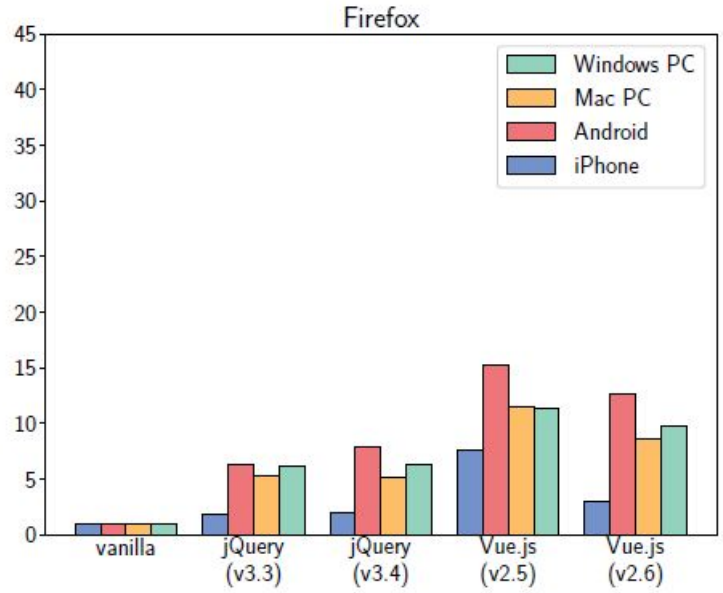

FIGURE 4.7 – Résultats tests performance sur mozilla firefox [24]

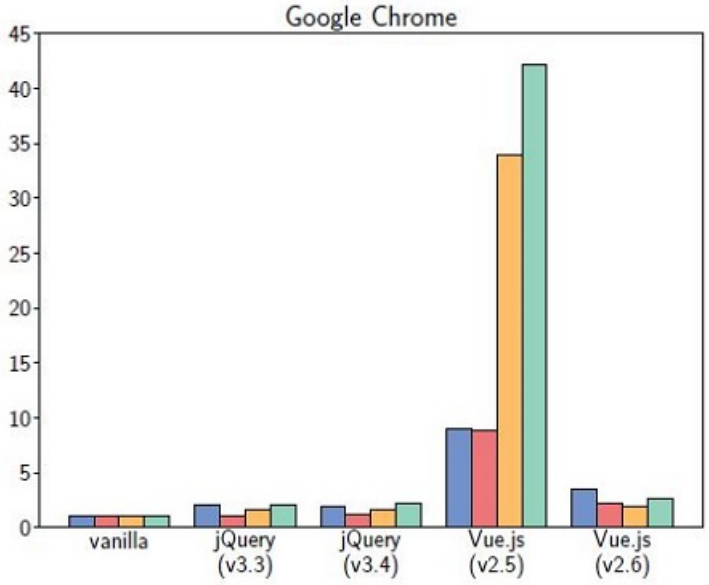

FIGURE 4.8 – Résultats tests performance sur google chrome [24]

Chapitre 5

Problématisation

5.1 Définition objectif à réaliser

Nous partons du postulat que :

— Il n'existe pas de réponse catégorique fiable et unanimement acceptée sur le meilleur framework Javascript à utiliser pour le développement d'une application web côté frontend

— Nous n'avons pas trouvé de comparaison utilisant les arbres de décision dans la littérature scientifique

Nous allons appliquer la méthode dite AHP, cette méthodologie est indiquée lors de la prise de décision entre plusieurs alternatives sur base de critères comparatifs, ce qui est notre cas. L'arbre résultant nous permettra de trier les frameworks par critère et en prenant en compte tous les critères.

Cette étude se développera en plusieurs étapes, d'abord nous présenterons la méthode AHP où nous expliquerons chaque étape de cette méthodologie , ensuite nous présenterons la méthodologie suivie pour sélectionner notre population de frameworks cible.

En troisième nous identifieront un ensemble de critères permettant de les départager sur base de notre révision de la littérature scientifique et de notre expérience d'utilisation de ces frameworks en développant pour chacune une application TodoList telle que conseillés dans la littérature scientifique [19, 29, 24]. En quatrième nous appliquerons la méthodologie AHP pour développer notre arbre de décision. Et finalement nous discuterons des résultats obtenus.

5.2 AHP : Analyse Hiérarchique des procédés

AHP ou analyse hiérarchique des procédés développée en 1980 par le mathématicien Thomas Saaty [32], est une méthode multicritère d'aide à la décision intégrant plusieurs critères arrivant à un choix justifié de technologie, la décision est alors dite rationnelle, systématique et correctement prise selon Gaëlle Guesdon, professeure à l'université de Laval au Québec [20]. Cette méthode permet de structurer des problèmes complexes en utilisant une forme visuellement compréhensible basée sur une hiérarchie de critères. Comme le rappelle Veena Nayak qui a fait un travail de recensement des articles scientifique promouvant

Echelle de valeurs (d'après Saaty, 1984)

Degrés d'importance	Définitions	Explications
1.0	importance égale des deux éléments	Deux éléments contribuent autant à la propriété
3.0	un élément est un peu plus important que l'autre	L'expérience et l'appréciation personnelles favorisent légèrement un élément par rapport à un autre
5.0	un élément est plus important que l'autre	L'expérience et l'appréciation personnelles favorisent fortement in élément par rapport à un autre
7.0	un élément est beaucoup plus important que l'autre	Un élément est fortement favorisé et sa dominance est attestée dans la pratique
9.0	un élément est absolument plus important que l'autre	Les preuves favorisant un élément par rapport à un autre sont aussi convaincantes que possible

FIGURE 5.1 – Échelle de valeurs (d'après Saaty, 1984) [20, 46]

les études utilisant les méthodes de comparaison de critères multiples, cette méthode est beaucoup utilisée dans le cadre de l'ingénierie de logiciel selon elle pour deux raisons : sa facilité d'apprentissage et sa simplicité d'implémentation [5].

La méthode se divise en plusieurs étapes :

— Identification des alternatives : on identifie les choix possibles
— Identification des critères : on identifie les critères pouvant les différencier
— Calcul poids critères : on calcule le poids des critères les uns par rapport aux autres, tous les critères ne se valant potentiellement pas, certains sont plus important que d'autres, cette étape nous permet de calculer la différence exacte
— Calcul poids des alternatives : on compare les critères les uns par rapport aux autres, puis on calcule le pourcentage menant au choix de l'alternative en prenant en compte les poids des critères

L'identification des alternatives se fait en choisissant une population de choix potentiels intéressants. L'identification des critères est réalisée sur base de la connaissance acquise de notre étude de la littérature scientifique et de notre expérimentation. Pour calculer les poids des critères, il faut remplir une matrice à deux dimensions permettant de comparer les critères les uns avec les autres deux à deux en précisant une échelle allant de $1/9$ jusqu'à 9 selon l'importance du paramètre (voir figure 5.1). Ces valeurs sont alors normalisées, la normalisation du résultat définit le poids de chaque critère. Finalement pour calculer le poids de chaque alternative, il faut les comparer selon la même logique que les critères et finalement normaliser chaque alternative en multipliant la valeur du critère par son poids [46].

Cette méthode nous fournit un pourcentage pour chaque alternative nous permettant de savoir quel choix faire.La comparaison étant réalisée par un être humain, il est possible que certaines valeurs soient inconsistantes. Cette métho-

Size of matrix (n)	**Random consistency index** (RI)
1	0
2	0
3	0.52
4	0.89
5	1.11
6	1.25
7	1.35
8	1.40
9	1.45
10	1.49

FIGURE 5.2 – AHP : Échelle de ratio de consistance admis par nombre de critères [46]

dologie règle ce problème en permettant de calculer un ratio de consistance, si ce CR est inférieur à 10 % on peut estimer que le résultat est consistant [32, 46].

Le ratio de consistance est obtenu grâce à cette formule :

$$CR = \frac{CI}{RI}$$

L'index randomisé (RI) est obtenu grâce à une table fournie par monsieur Saaty (voir figure 5.2), comme nous disposons de 5 critères, RI vaut dans notre cas 1.11, l'index de consistance (CI) est obtenu en suivant plusieurs étapes intermédiaires.

D'abord nous devons calculer un nouveau vecteur de priorité, pour ce faire nous devons multiplier la matrice 5x5 de réciprocité par la matrice 5x1 contenant les poids de chaque critère pour obtenir les nouveaux poids. Ensuite nous devons diviser chaque nouveau poids par l'ancien poids équivalent, nous additionnons toutes ces nouvelles valeurs que nous divisons pour obtenir la moyenne maximale des poids (λ max) tel que défini par cette formule (voir figure 5.27) :

$$\lambda max = \frac{1}{n} \sum_{i=1}^{n} \frac{NVi}{PVi}$$

Nous pouvons ensuite obtenir CI suivant cette formule (n étant le nombre de critères) :

$$CI = \frac{(\lambda max - n)}{n - 1}$$

5.3 Population frameworks ciblés

5.3.1 Liste de base

Nous avons décidé d'utiliser cette page récapitulative reprise sur Github pour démarrer notre sélection : https://github.com/collections/front-end-javascript-frameworks
Nous avons choisi cette plateforme car elle héberge les projets open source les plus populaires. Nous avons ensuite repris et agréger les informations importantes dans un tableau Excel.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Frameworks	Derniere année maj	Année création	Nb étoiles	Watchers	Forks	License	Derniere version	Date dern release	Support	Site dédié	Documentation	Lien
2	seemple	2020	2013	810	52	82	MIT	2.4.18	20/01/2020	/	oui	oui	https://github.com/finom/seemple
3	Aurelia2	2021	2018	965	68	86	MIT	0.6	18/12/2019	Gratuit et payant sur leur site	oui	oui	https://github.com/aurelia/aurelia
4	neo	2021	2019	1200	24	60	MIT	1.4.56	2/02/2021	slack	oui	oui	https://github.com/neomjs/neo
5	dojo	2020	2007	1400	167	534	BSD License	1.16.3	13/06/2020	Non	oui	oui	https://github.com/dojo/dojo
6	nuclearjs	2018	2014	2200	206	158	MIT	1.4	21/09/2016	/	oui	oui	https://github.com/optimizely/nuclear-js
7	material	2020	2015	3100	191	745	MIT	4.1.1	2/05/2018	/	oui	oui	https://github.com/Daemonite/material
8	spine	2017	2011	3600	113	448	MIT	1.6.2	5/02/2016	Google group	oui	oui	https://github.com/spine/spine
9	Litelement	2021	2017	4100	167	321	MIT	2.4	19/08/2020	/	oui	oui	https://github.com/Polymer/lit-element
10	knockout js	2020	2014	9900	558	1600	MIT	3.5.1	5/11/2019	/	oui	oui	https://github.com/knockout/knockout
11	Marko	2021	2015	10000	233	603	MIT	5.1.10	5/02/2021	Discord, stackoverflow, twitter	oui	oui	https://github.com/marko-js/marko
12	aurelia	2021	2014	11600	451	684	MIT	1.3.1	29/01/2019	Gratuit et payant sur leur site	oui	oui	https://github.com/aurelia/framework
13	Mithirljs	2020	2014	12600	332	937	MIT	1.1.7	24/09/2019	/	oui	oui	https://github.com/MithrilJS/mithril.js
14	riot	2021	2013	14400	423	1000	MIT	5.2.0	1/02/2021	/	oui	oui	https://github.com/riot/riot
15	hyperapp	2021	2016	18300	349	814	MIT	2.0.12	27/01/2021	Discord	non	oui	https://github.com/jorgebucaran/hyperapp
16	polymer	2021	2015	21500	926	2000	BSD License	3.4.1	30/04/2020	/	oui	oui	https://github.com/Polymer/polymer
17	ember	2021	2011	21700	953	4200	MIT	3.25	9/02/2021	/	oui	oui	https://github.com/emberjs/ember.js
18	todomvc	2020	2012	27200	1200	13900	MIT	1.3.1	23/08/2018	/	oui	oui	https://github.com/tastejs/todomvc
19	backbone	2020	2010	27700	1400	5600	MIT	1.4	19/02/2019	/	oui	oui	https://github.com/jashkenas/backbone
20	preact	2021	2015	28300	439	1600	MIT	10.5.12	27/01/2021	/	oui	oui	https://github.com/preactjs/preact
21	svelte	2021	2016	43700	868	2000	MIT	3.32.2	8/02/2021	/	oui	oui	https://github.com/sveltejs/svelte
22	angular	2021	2014	70600	3200	18500	MIT	11.1.1	28/01/2021	Gitter, Meetup	oui	oui	https://github.com/angular/angular
23	react	2021	2013	163000	6700	32700	MIT	17.0	20/10/2020	/	oui	oui	https://github.com/facebook/react
24	vue	2021	2016	179000	6300	28100	MIT	2.6.11	20/08/2020	/	oui	oui	https://github.com/vuejs/vue

Liste de base des frameworks cibles

5.3.2 Filtrage

La liste initiale contenant 23 frameworks, nous avons pris la décision de la réduire à 5 frameworks finaux, ce nombre nous semble suffisant afin de démontrer l'utilisabilité de notre arbre. Contrairement aux articles étudiés précédemment [29, 31, 30, 11], nous aurons une démarche scientifique dans la sélection des frameworks comparés, nous filtrerons sur base de critères clairement exprimés qui nous permettront de justifier notre choix. Tout d'abord nous avons décidé de ne garder que les frameworks qui sont encore maintenus, nous définissons un framework comme à jour s'il a eu au moins une mise à jour durant cette année (2021). Ce filtrage nous a permis de nous séparer de ces frameworks : Seemple, Dojo, Nuclearjs, Material, Spine, KnockoutJs, MithrilJs, TodoMvc, Backbone.
Ensuite pour renforcer la pertinence de nos frameworks choisis, nous nous sommes séparés des frameworks qui ne sont pas matures, nous définissons un framework mature comme un framework qui a au moins 5 ans d'existence, cette longévité nous semble suffisante pour atténuer au maximum le risque que la liste finale soit obsolète dans les années proches qui suivront la publication des résultats de notre étude. Ce filtrage nous a permis de nous séparer de ces frameworks : Aurelia2, Neo, LitElement.
Finalement après lecture approfondie des pages de documentations fournies pour les frameworks restants, nous avons éliminé les frameworks vivement déconseillés par les auteurs. Ce filtrage nous a permis de nous séparer du framework : Polymer [9].
Nous avons décidé d'utiliser un critère cité et exploité dans plusieurs articles étudiés précédemment [19, 28] que nous résumons par l'engouement. Nous simplifions ce concept en le basant sur le nombre d'étoiles attribué à un projet (un compte github ne peut voter qu'une seule fois). Nous avons dès lors trié de manière décroissante les 10 frameworks restants sur base du nombre d'étoiles reçues.
Nous avons sélectionné les 5 premiers frameworks qui génèrent d'après notre définition le plus d'engouement, dont voici la liste :

— Vue
— React
— Angular
— Svelte,
— Preact

5.3.3 Présentation succincte de chaque frameworks

Svelte

Svelte est un framework conçu en 2016 par Rich Harris qui est un développeur indépendant, il est maintenu par le fondateur et une équipe de bénévoles actuellement [42]. Svelte est écrit en Typescript. Tout comme la plupart de ses concurrents Svelte permet de créer une application web en déclarant des composants, la différence est qu'il propose un compilateur qui se charge d'optimiser les modifications du DOM plus précisément que la concurrence d'après le fondateur [42]. Les créateurs avancent plusieurs avantages à l'utilisation de leur framework [42]. D'abord il est conçu de sorte à réduire au maximum l'écriture de code afin d'améliorer la lisibilité du code et d'après eux le taux potentiel

de bugs augmentant avec la taille du code, leur solution permet de créer des applications plus robustes car plus faciles à tester [42]. Le compilateur se charge d'optimiser les manipulations du DOM pour le programmeur de manière automatisée [42]. En deuxième pour se démarquer de la concurrence, ils ont adopté une autre approche de rafraîchissement plus optimisé du DOM en évitant le DOM virtuel implémenté par la plupart des autres frameworks [42]. En effet le DOM virtuel se charge d'optimiser uniquement les éléments strictement nécessaires d'une page lors d'un changement d'état mais rafraîchit quand même des éléments non nécessaires d'après Rich Harris [42]. L'idée ici est de modifier uniquement les valeurs dynamiques, c'est rendu possible car le compilateur sait au temps de la compilation quels états peuvent changer et quelles sont les valeurs possibles (ce qui n'est pas le cas pour les autres frameworks) [42]. En troisième, Svelte est plus permissif et offre plus de liberté que ses concurrents car il n'impose pas d'avoir une seule racine dans l'arbre Html, on peut en avoir autant qu'on le désire [42]. Finalement ils proposent une version simplifiée de gestion d'état des valeurs dynamiques injectées dans les pages web [42]. Cependant ce framework a quelques désavantages, il n'est soutenu par aucun acteur majeur des entreprises engagées dans le web, la communauté est plus restreinte que la concurrence, ce qui a un impact sur plusieurs aspects tels que les réponses aux questions éventuelles qui peuvent prendre plus de temps et une offre de package réduite [40].

Angular

Angular est un framework écrit en Typescript créé en 2014 par une équipe chez Google. Il est actuellement maintenu par un groupe de bénévole et des employés de Google. il est également soutenu financièrement par Google [1]. Ce framework est un précurseur et l'un des premiers frameworks à avoir offert la double liaison des variables avec le DOM [1]. Il offre plusieurs avantages, il se définit comme portable, son ambition est de pouvoir être utilisé pour développer des applications sur n'importe quelle plateforme pouvant héberger des applications web [1]. Énormément d'outils ont été développés et mis à disposition de la communauté gratuitement pour aider les développeurs, des plugins pour les IDE et les navigateurs, des packages permettant d'éviter de recréer la roue sont disponibles via le gestionnaire de package NPM, un CLI a également été mis à disposition pour faciliter le démarrage d'un projet [1]. Ils font régulièrement des mises à jour pour rajouter des nouvelles fonctionnalités ou corriger des bugs [1]. Il y a une énorme communauté, beaucoup de dynamisme dans les forums (Reddit, Stackoverflow, ..), on a une réponse très rapidement à des questions, il n'est pas dur de trouver de l'aide lorsqu'on est bloqué sur un développement [1]. Il a néanmoins plusieurs désavantages, il impose l'utilisation du Typescript, ce qui augmente la courbe d'apprentissage du framework et ralentit l'initialisation d'un projet [1]. Aucune rétrocompatibilité avec la 1ère version d'Angular à imposer aux développeurs de réécrire presque complètement leurs applications lors du passage à la version 2 [1]. Ils avancent la performance comme étant l'un des avantages majeurs mais la plupart des concurrents ont été construits après Angular avec l'optique d'offrir des performances plus rapides [42, 39, 37, 45]. Comme dit dans l'article étudié plus tôt, Angular impose la compréhension de plusieurs concepts qui ne sont pas toujours bien assimilés par les développeurs, ce qui les amènent régulièrement à ne pas utiliser les solutions les plus efficientes

à leurs problèmes [30].

React

ReactJs est une librairie qui a été conçue en 2013 par Jordan Walke, un employé de Facebook, Facebook continue à maintenir le projet avec des développeurs bénévoles indépendants [54]. Les créateurs définissent ReactJs comme une librairie permettant de créer des interfaces utilisateurs performantes, facile à maintenir et plus sûr [39, 51]. L'atout majeur de cette librairie est la mise en place d'une couche intermédiaire dans les opérations de modifications du DOM (DOM virtuel). Celle-ci se chargeant d'optimiser les opérations et de modifier le minimum d'éléments requis de manière automatique, lors d'une modification quelconque le développeur n'a plus qu'à demander à faire un rendu complet et le DOM se chargera de rafraîchir les éléments nécessaires sur l'interface utilisateur. Ils offrent également un parser optionnel appelé "JSX" permettant de mélanger Html et Javascript afin d'optimiser et réduire les concaténations et manipulation du DOM sous forme de chaîne de caractères. La librairie permet de créer des composants qu'il suffit d'assembler, il est également possible d'importer des blocs développés par d'autres développeurs, ce qui pousse à la réutilisation de code. React promeut également la programmation fonctionnelle afin d'avoir un meilleur contrôle sur le résultat des opérations sur les objets visés et pouvoir développer du code testable et fiable. Le principal défaut de cette librairie est la courbe d'apprentissage qui semble être trop élevé entre les nouveaux concepts (JSX, virtual DOM, component, API de base, ...) [27]. Cette librairie a été citée et comparée plusieurs fois dans plusieurs articles étudiés précédemment, il a provoqué encore récemment un énorme engouement. [56, 11].

Preact

Preact est une librairie alternative à ReactJs qui a été conçue par une équipe différente en 2015 pour combler plusieurs points faibles de ReactJs : un DOM virtuel alléger et moins complexe, une taille globale de la librairie optimisée et plus légère, portable (utilisable pour n'importe quel type de projet très facilement) [37]. On retrouve généralement deux points faibles : la taille de la communauté qui est plus faible que ReactJs et l'API étant moins fourni pour optimiser la taille (et donc la performance) n'offre pas certaines fonctionnalités qui peuvent être jugées comme indispensables (ex : context, propTypes, createClass, ...). [44]. Cette librairie n'avait pas encore été citée dans la littérature scientifique.

Vue

Vue a été créé par Evan You, un ancien employé de Google en 2014. L'idée était d'extraire les fonctionnalités qui lui semblait d'après lui indispensable d'Angular dans une version plus allégée. Le framework est actuellement maintenu par lui et un groupe de bénévole [45]. Ce framework offre plusieurs avantages, d'abord la taille du framework est fortement réduite ce qui diminue la taille de la bande passante nécessaire pour charger la librairie et génère donc un gain en performances [45]. En deuxième les créateurs définissent ce framework comme étant "progressif", on peut l'utiliser sur une simple page générée par le

backend afin de dynamiser le contenu d'une page ou on peut créer une application complète [45]. En troisième, il offre un DOM virtuel comme ses concurrents qui permet d'optimiser automatiquement les manipulations du DOM afin de rafraîchir uniquement les parties nécessaires de l'écran [45]. En quatrième, il existe plusieurs outils pour aider les développeurs à créer leur application (CLI) ou investiguer et corriger des bugs (plugins du navigateur) [43]. Finalement, ce framework n'impose pas l'apprentissage de technologies/languages intermédiaires (JSX, Typescript, ...), les concepts sont plus simples, ce qui réduits la courbe d'apprentissage [43]. Ce framework a néanmoins quelques désavantages, il serait implémenté par des leaders du marché chinois (Xiaomi, Alibaba), dès lors bien que la documentation de base soit bien fournies dans des langues accessibles, une partie non négligeable des packages et de la documentation ne sont pas accessibles aux développeurs vivant en dehors de la Chine [43]. Il n'est soutenu financièrement par aucun leader du marché, ce qui peut freiner son adoption dans la création d'application très complexe par des organismes qui cherchent avant tout de la fiabilité, du support et de la stabilité [43]. Le nombre de packages/plugins n'est pas négligeable mais reste réduit comparé à la concurrence [43].

5.4 Critères de comparaison

5.4.1 TodoList

Présentation fonctionnelle

Comme conseillés dans plusieurs articles étudiés dans la littérature scientifique, nous allons développer une application TodoList pour chacun des frameworks mentionnés plus haut [19, 29, 24].

Une application TodoList permet à son utilisateur de gérer une liste de tâche à réaliser dans le futur, une tâche consiste en un texte et un status (ouvert/finalisé) (voir figure 5.3). Il pourra insérer autant de nouvelles tâches qu'il le désire et les éditer si nécessaire, il pourra également changer leur status (ouvert -> finalisé, finalisé -> ouvert), il lui sera également possible de changer le status de toutes les tâches vers une valeur en un seul clique si nécessaire. Pour améliorer la visualisation, il pourra également filtrer la liste de tâche par leur status. Finalement les tâches seront stockées dans une base de données spéciale du navigateur (local storage) afin que l'utilisateur puisse les retrouver par la suite (par simple rafraîchissement de la page ou en fermant et rouvrant le navigateur).

Détails techniques

Nous utiliserons Git comme gestionnaire de code pour stocker le code source de nos 5 applications. L'url : `https://github.com/aliziani/Thesis_compare_frameworks`

Nous allons reprendre un design complet provenant d'un framework présent dans la liste de base mais non retenu pour ne pas recréer la roue et parce qu'il remplit nos exigences fonctionnelles [47]. Certains des frameworks dans notre liste sont déjà exemplifiés par TodoMvc mais ont été implémentés avec une version antérieure qui n'est plus maintenue et des éléments dépréciés. Nous

FIGURE 5.3 – Exemple design [47]

reprendrons ce qui peut être repris et adapterons le code pour qu'il soit à jour avec la dernière version disponible du framework.

Les versions utilisées :

— Angular : 11.0.9
— Vue : 3.0.11
— React : 17.0.2
— Preact : 10.5.13
— Svelte : 3.0.0

En ce qui concerne notre environnement de développement, nous utiliserons "Visual Studio Code" comme interface de développement intégré. Nous utiliserons exclusivement Google Chrome (v90.0.4430.212). Le développement d'une application frontend implique l'utilisation d'autres technologies, presque la totalité des frameworks retenus nécessitent d'avoir installé NodeJs et Npm au préalable. NodeJs est un service http développé en Javascript qui permet d'héberger un service backend, il est utilisé comme service http [25]. Npm est un programme tournant sur NodeJs qui fait office de gestionnaire de package, il permet de récupérer les dépendances d'un projet [26].

5.4.2 Liste complète

Sur base de notre étude et de notre expérimentation nous avons défini une liste de critères permettant de départager la liste de frameworks sélectionnés.

L'effort d'apprentissage

Nous désirons représenter ici à quel point la barrière d'apprentissage est élevé pour pouvoir commencer à travailler avec ce framework, plus bas vous trouverez des éléments non exclusifs que nous considérons faire partie de ce critère.

- La documentation : nous nous concentrerons plus précisément sur plusieurs points, tel que le multilinguisme du contenu, la conception du site de documentation favorise-t-elle l'accès à tous (mise en place des composants d'accessibilité [7]), la disponibilité d'accès à des vidéos (youtube ou auto-hébergée référencé par leur site directement), des outils d'apprentissage en temps réel, des tutoriels pas-à-pas seront pris en compte pour comparer les frameworks entre eux
- Langages prérequis : est-ce que des langages intermédiaires sont imposés pour pouvoir utiliser le framework. (ex : Typescript)
- Outils prérequis : est-ce que des logiciels ou des outils sont nécessaires avant de pouvoir utiliser le framework. (ex : NodeJs, CLI,...)

L'effort de développement

Nous désirons représenter ici la difficulté rencontrée par un développeur pour pouvoir développer une application avec un framework, plus bas vous trouverez des éléments non exclusifs que nous considérons faire partie de ce critère.

- CLI : ce critère nous permettra de distinguer les frameworks sur la mise à disposition ou non d'une interface de commande en ligne facilitant et automatisant toutes les étapes de développement du logiciel désiré (initialisation, mise en place environnement de test, ajout de composants, exécution de scripts personnalisés, ...)
- Plugins navigateurs : est-ce qu'il existe des plugins facilitant la vie des développeurs dédiés et/ou spécialisé à notre liste de frameworks. Nous nous limiterons à Google Chrome, nous récupérerons cette information via son magasin de plugins
- Le nombre de questions sur Stackoverflow : Stackoverflow est un forum dédié à différents sujets, plusieurs concerne des sujets concernant la programmation, permettant aux programmeurs de poser des questions et d'y répondre, c'est une mine d'or pour les développeurs, avoir des questions déjà posées sur ce forum peut faire économiser un temps très précieux sur des questions d'implémentation de concepts et des résolutions de bug. Nous récupérerons ce nombre via la fonctionnalité de recherche fournie par le site, une question peut se voir attribuer plusieurs tags, un framework peut être orthographié différemment et avoir des synonymes, pour cette raison nous prendrons le nombre maximal identifié
- Temps d'initialisation : corresponds au temps d'initialisation d'un projet (installation avec dépendances), celui-ci sera récupéré de notre expérience du développement de l'application TodoList
- Temps pour démarrer le projet TodoList : nous mentionnerons le temps que prends l'environnement de développement pour démarrer et permettre au développeur de tester son code
- Temps de prise en main : ce dernier critère correspondra au temps que nous avons pris pour implémenter notre application TodoList

Stabilité

Nous désirons représenter ici la fiabilité de l'écosystème autour du framework afin de rassurer un développeur d'utiliser ou baser son application sur ce framework, plus bas vous trouverez des éléments non exclusifs que nous considérons faire partie de ce critère.

- Nombre de dépendances du framework : un framework est un assemblage de librairie, avoir énormément de dépendance peut ouvrir la voie à des complications en terme de maintenance (régression, obsolète, failles sécurité, ...). Cette information sera récupérée sur le repository même dans Github
- Changelog : une release peut s'accompagner de changement structurel important, il est toujours préférable de pouvoir connaître à l'avance le contenu d'une release pour pouvoir évaluer la faisabilité en temps et coût d'une mise à jour. Cette information sera récupérée à partir de Github
- Financement : ce critère nous permettra de savoir comment est maintenu le framework, est-ce un leader du marché qui dédie une partie de son effectif pour son maintien ou est-ce que l'équipe est entièrement bénévole. Nous récupérerons cette information sur leurs sites officiels respectifs. Nous partirons du principe qu'un framework maintenu que par des bénévoles sera discriminante car aucune garantie n'est fourni sur la résolution de bugs éventuels ou l'ajout de fonctionnalités
- Outils de mise à jour : ce critère nous permettra de distinguer les frameworks offrant des outils d'aide à la mise à jour (ex : comparatifs de versions éloignées, explications détaillées des étapes à suivre, ...)
- Année de création : l'âge d'un framework n'est pas directement proportionnel à un bon niveau de stabilité mais un framework créer récemment peut manquer de maturité

Attractivité

Nous étions arrivé à la conclusion qu'il existait un effet de mode sur la sélection des frameworks [31, 30, 11], nous regroupons ici des éléments non exclusifs permettant d'évaluer l'attractivité d'un framework pouvant mener à son choix.

- Nombre étoiles sur github : un compte utilisateur ne peut placer qu'une étoile par projet, nous sommes conscients que le nombre d'étoiles ne permet pas de connaître le nombre de personnes ayant placé le framework dans leurs favoris car il est possible d'avoir plusieurs comptes par personne, néanmoins ça reste un bon indicateur de notre point de vue de la popularité du framework
- Nombre téléchargements : nombre de téléchargements uniques de la librairie principale de chaque framework. Nous récupérerons cette information via le site NPM [26]
- Nombre total de pages référençant le framework : nous reprendrons ici le nombre total retourné par Google en utilisant la page d'accueil
- Classement satisfaction : une enquête a été réalisé sur l'année 2020 sur une population de 23 765 personnes concernant leurs préférences concernant la satisfaction des frameworks Javascript, nous n'avons pas trouvé pareil enquête dans la littérature scientifique, il nous semble important

de pouvoir réutiliser cette information au vu de la quantité de personnes questionnées [41]

Performance

Nous regroupons ici des éléments non exclusifs permettant de se faire une idée sur le niveau de performance fourni par les frameworks.

- Temps chargement page TodoList : nous reprendrons ici le temps de chargement de la page principale de l'application TodoList développée sur chaque framework
- Poids framework : ce critère nous permettra de comparer le poids du framework en kb
- Note globale outils Lighthouse sur notre application TodoList : nous avons pris la décision de sélectionner l'outils LightHouse [18], pour différentes raisons, d'abord il y a plusieurs recherches qui promeuvent l'utilisation d'outils automatiques d'évaluation de performance [10, 24], en deuxième Google Chrome est le navigateur le plus populaire au monde [35] et offre cet outil par défaut dans son navigateur ce qui est très pratique car nous avons utilisé exclusivement ce navigateur, finalement parce que cet outil exécute une batterie de tests complets [12, 13, 14, 15, 16, 17]

5.5 Évaluation

5.5.1 Identification des alternatives

La première étape consistait à rassembler une liste d'alternatives :

- React
- Vue
- Svelte
- Angular
- Preact

5.5.2 Identification des critères

La seconde étape consistait à rassembler une liste de critères pouvant les départager :

- Effort apprentissage (L)
- Effort développement (D)
- Stabilité (S)
- Attractivité (A)
- Performance (P)

5.5.3 Calcul poids des alternatives

Pour pouvoir calculer le poids de chaque framework pour chaque critère, nous allons centraliser dans un tableau la récolte d'informations concernant les éléments constitutifs identifiés précédemment. Pour avoir une vue simplifiée, nous appliquerons un code couleur, nous utiliserons le vert pour le meilleur résultat, le rouge pour le pire résultat, et l'orange pour les résultats intermédiaires.

Effort apprentissage	React	Vue	Svelte	Angular	Preact
Offre documentation	Oui (mais pas de vidéos)	Oui complet	Oui (mais pas multilingue)	Oui (pas de vidéos et pas multilingue)	Oui (pas de vidéos et pas multilingue)
Est-ce que le framework impose des outils supplémentaires ?	Oui	Oui	Non	Oui	Non
Est-ce que le framework impose des langages supplémentaires ?	Non (optionnel)	Non	Non	Oui	Non (optionnel)

FIGURE 5.4 – AHP : résultat analyse effort apprentissage pour chaque framework

Effort Apprentissage	React	Vue	Svelte	Angular	Preact
React	1,00	0,50	0,33	4,00	0,33
Vue	2,00	1,00	0,33	5,00	0,25
Svelte	3,00	3,00	1,00	7,00	2,00
Angular	0,25	0,20	0,14	1,00	0,17
Preact	3,00	4,00	0,50	6,00	1,00
Total	9,25	8,70	2,31	23,00	3,75

FIGURE 5.5 – AHP : réciprocité pour le critère effort apprentissage

Nous nous baserons sur ce tableau pour comparer les frameworks entre eux afin d'identifier les degrés d'importance des uns par rapport aux autres.

L'effort d'apprentissage

La compilation des informations concernant ce critère est consultable via cette figure 5.4. Sur base de ce tableau nous construisons une table de réciprocité où nous comparons les frameworks les uns avec les autres en attribuant pour chaque pair un degré d'importance allant de 1 à 9 (voir figure 5.1), voir figure 5.5.

Nous pouvons remarquer que la seule différence entre React et Vue concerne la mise à disposition de vidéos éducatives fournies par l'un et pas par l'autre, nous décidons dès lors de définir un degré intermédiaire 2 au profit de Vue car il existe une différence au profit de Vue mais elle reste négligeable. En comparant React et Svelte, nous remarquons une différence non négligeable, React nécessite la mise en place de plusieurs outils intermédiaire alors que Svelte non, nous considérons dès lors que Svelte est plus facile à utiliser que React, nous attribuons un degré 3 au profit de Svelte. En comparant React et Angular, nous remarquons que Angular nécessite plus d'efforts car ce framework impose un langage intermédiaire (Typescript), nous observons aussi qu'il n'offre pas de documentation multilingue, pour ces raisons nous attribuons le degré intermédiaire 4 au profit de React. En comparant React et Preact, nous observons que Preact ne nécessite pas d'outils intermédiaires, mais n'offre pas de documentation dans plusieurs langues, pour ces raisons nous attribuons un degré 3 au profit de Preact. En comparant Vue et Svelte, nous observons que Vue offre la meilleure documentation mais impose l'utilisation d'outils supplémentaires, pour ces raisons nous attribuons un degré 3 au profit de Svelte. En comparant Vue et Angular, nous observons une différence de documentation mais elle reste négligeable par contre Vue n'impose pas de langages supplémentaires, pour ces raisons nous attribuons le degré 5 au profit de Vue. En comparant Vue et Preact, nous observons également une différence de documentation qui est négligeable mais Preact n'impose aucun intermédiaire, pour ces raisons nous attribuons le degré 4 au profit de Preact. En comparant Svelte et Angular, nous remarquons que Svelte offre un climat d'apprentissage plus facile que Angular car il

Effort Apprentissage	React	Vue	Svelte	Angular	Preact	Total par ligne	Vecteur priorité (poids)
React	0,108	0,057	0,144	0,174	0,089	0,573	0,115
Vue	0,216	0,115	0,144	0,217	0,067	0,760	0,152
Svelte	0,324	0,345	0,433	0,304	0,533	1,940	0,388
Angular	0,027	0,023	0,062	0,043	0,044	0,200	0,040
Preact	0,324	0,460	0,216	0,261	0,267	1,528	0,306
Total	1,00	1,00	1,00	1,00	1,00	5,00	1,00

FIGURE 5.6 – AHP : normalisation et poids pour le critère effort apprentissage

Effort Apprentissage	React	Vue	Svelte	Angular	Preact	Vecteur priorité (VP)	Nouveau vecteur priorité (VR)	VR/VP	RC	
React	1,00	0,50	0,33	4,00	0,33	0,115	0,580	5,064	CI	0,06437053
Vue	2,00	1,00	0,33	5,00	0,25	0,152	0,787	5,178		
Svelte	3,00	3,00	1,00	7,00	2,00	0,388	2,081	5,364	RI	1,11
Angular	0,25	0,20	0,14	1,00	0,17	0,040	0,205	5,143		
Preact	3,00	4,00	0,50	6,00	1,00	0,306	1,693	5,539		
							Somme	26,287	CR	0,05799147
							Lambda max	5,257		

FIGURE 5.7 – AHP : CR pour le critère effort apprentissage

n'impose aucun intermédiaire et offre également des vidéos éducatives dans sa documentation, nous attribuons un degré 7 au profit de Svelte. En comparant Svelte et Preact, nous remarquons uniquement une différence d'offre sur la documentation qui reste négligeable, pour cette raison nous attribuons un degré intermédiaire 2 au profit de Svelte. Finalement en comparant Preact et Angular, nous remarquons que la documentation de Preact est légèrement moins fournie que Angular, mais par contre elle n'impose aucun intermédiaire, nous attribuons un degré 6 au profit de Preact.

Ensuite nous normalisons les valeurs obtenues dans chaque cellule afin de calculer leur total et finalement leur poids (appelé aussi vecteur de priorité) voir figure 5.6.

Finalement nous calculons le CR (voir figure 5.7), ce qui nous donne une valeur de 0.06, 0.06 étant inférieur à 0.1, ceci indique que les poids définis sont acceptables et offrent un bon niveau de consistance.

L'effort de développement

La compilation des informations concernant ce critère est consultable via cette figure 5.8. Sur base de ce tableau nous construisons une table de réciprocité où nous comparons les frameworks les uns avec les autres en attribuant pour chaque pair un degré d'importance allant de 1 à 9 (voir figure 5.1), voir figure 5.9.

En comparant React et Vue, on observe que Stackoverflow contient plus de questions référençant React que Vue, les différences de temps d'initialisation et de démarrage de l'environnement de développement sont négligeables par contre on peut remarquer que créer l'application TodoList a pris deux fois plus

Effort développement	React	Vue	Svelte	Angular	Preact
CLI disponible	Oui	Oui	Non	Oui	Oui
Plugins navigateurs	Oui	Oui	Oui	Oui	Oui
Nombre de questions sur stackoverflow	305.067	78.336	1792	262.169	358
Temps pour initialiser le projet todolist (min)	20	15	0,5	20	5
Temps pour démarrer le projet todolist (seconde)	65	70	5	95	30
Temps développement todolist (heure)	8	4	3	5	8

FIGURE 5.8 – AHP : résultat analyse effort de développement pour chaque framework

Effort Développement	React	Vue	Svelte	Angular	Preact
React	1,00	0,33	0,20	3,00	0,25
Vue	3,00	1,00	0,25	3,00	0,33
Svelte	5,00	4,00	1,00	6,00	4,00
Angular	0,33	0,33	0,17	1,00	0,33
Preact	4,00	3,00	0,25	3,00	1,00
Total	13,33	8,67	1,87	16,00	5,92

FIGURE 5.9 – AHP : réciprocité pour le critère d'effort de développement

de temps avec React comparé à Vue, pour ces raisons nous attribuons le degré 3 au profit de Vue. En comparant Svelte et React, nous observons que Svelte ne fournit pas de CLI pour faciliter le développement, Stackoverflow contient moins de questions pour Vue que pour React par contre les métriques concernant l'application TodoList sont toutes en faveur de Vue, pour ces raisons nous attribuons le degré 5 au profit de Svelte. En comparant Angular et React, nous n'observons pas de différences très importante mise à part le temps de développement de l'application TodoList qui vaut un peu moins du double pour React, pour ces raisons nous attribuons le degré 3 au profit de React. En comparant React et Preact, on observe plusieurs différences, le nombre de questions dédiées à Preact est moindre par rapport à React, toutes les métriques concernant l'application TodoList sont en faveur de Preact, pour ces raisons nous attribuons le degré intermédiaire 4 au profit de Preact. En comparant Vue et Svelte, nous observons que Svelte ne fournit pas de CLI et Stackoverflow offre moins de questions que Vue par contre toutes les métriques de l'application TodoList sont en faveur de Svelte, pour ces raisons nous attribuons le degré 4 au profit de Svelte. En comparant Vue et Angular, nous observons que la seule réelle différence est le nombre de questions sur Stackoverflow, le reste des différences sont négligeables, pour ces raisons nous attribuons le degré 3 au profit de Vue. En comparant Vue et Preact, nous remarquons que Stackoverflow offre plus de questions pour Vue mais les métriques concernant l'application TodoList sont en faveur de Preact, pour ces raisons nous attribuons le degré 3 à Preact. En comparant Svelte et Angular, nous observons que Svelte n'offre pas de CLI et toutes les métriques concernant l'application TodoList sont en faveur de Svelte néanmoins Stackoverflow offre plus de question pour le framework Angular, pour ces raisons nous attribuons le degré 6 au profit de Svelte. En comparant Svelte et Preact, nous observons que Svelte n'offre pas de CLI contrairement à Preact, les métriques concernant l'application TodoList sont en faveur de Svelte et Stackoverflow contient plus de questions concernant le framework Svelte, pour ces raisons nous attribuons le degré 4 au profit de Svelte. Finalement en comparant Angular et Preact, nous observons que Stackoverflow offre plus de questions pour le framework Angular mais toutes les métriques concernant l'application TodoList sont en faveur d'Angular, pour ces raisons nous attribuons un degré de 3 au profit de Preact.

Ensuite nous normalisons les valeurs obtenues dans chaque cellule afin de calculer leur total et finalement leur poids (appelé aussi vecteur de priorité) voir figure 5.10.

Finalement nous calculons le CR (voir figure 5.11), ce qui nous donne une

Effort Développement	React	Vue	Svelte	Angular	Preact	Total par ligne	Vecteur priorité (poids)
React	0,075	0,038	0,107	0,188	0,042	0,450	0,090
Vue	0,225	0,115	0,134	0,188	0,056	0,718	0,144
Svelte	0,375	0,462	0,536	0,375	0,676	2,423	0,485
Angular	0,025	0,038	0,089	0,063	0,056	0,272	0,054
Preact	0,300	0,346	0,134	0,188	0,169	1,137	0,227
Total	1,00	1,00	1,00	1,00	1,00	5,00	1,00

FIGURE 5.10 – AHP : normalisation et poids pour le critère effort de développement

Effort Développement	React	Vue	Svelte	Angular	Preact	Vecteur prio	Nouveau vecteur priorité (VR)	VR/VP	RC	
React	1,00	0,33	0,20	3,00	0,25	0,090	0,455	5,048	CI	0,10722984
Vue	3,00	1,00	0,25	3,00	0,33	0,144	0,774	5,387		
Svelte	5,00	4,00	1,00	6,00	4,00	0,485	2,745	5,663	RI	1,11
Angular	0,33	0,33	0,17	1,00	0,33	0,054	0,289	5,316		
Preact	4,00	3,00	0,25	3,00	1,00	0,227	1,303	5,730		
							Somme	27,145	CR	0,09660346
							Lambda max	5,429		

FIGURE 5.11 – AHP : CR pour le critère effort de développement

valeur de 0.097, 0.097 étant inférieur à 0.1, ceci indique que les poids définis sont acceptables et offrent un bon niveau de consistance.

Stabilité

La compilation des informations concernant ce critère est consultable via cette figure 5.12. Sur base de ce tableau nous construisons une table de réciprocité où nous comparons les frameworks les uns avec les autres en attribuant pour chaque pair un degré d'importance allant de 1 à 9 (voir figure 5.1), voir figure 5.13.

En comparant React et Vue, on observe que React est soutenu financièrement et est le framework qui a la plus longue longévité, pour ces raisons nous attribuons le degré 7 au profit de React. En comparant React et Svelte, nous avons les mêmes observations que pour Vue à l'exception que Svelte a le moins de dépendances externe, pour ces raisons nous attribuons le degré 6 au profit de React. En comparant React et Angular, nous observons qu'ils sont tous deux soutenus financièrement par deux grandes entreprises présentes sur le web (Facebook, Google), leur différence de longévité est négligeable, par contre le nombre de dépendance est plus grand pour Angular mais il offre un outil de mise d'aide à la mise à jour [34], pour ces raisons nous attribuons le degré 2 au profit de React. En comparant React et Preact, nous observons que Preact est maintenu bénévolement, que la différence de longévité est moins négligeable, qu'il a moins de dépendances mais qu'il fournit quand même une aide à la mise à jour [38], pour ces raisons nous attribuons le degré 4 au profit de React. En comparant Vue et Svelte, on observe uniquement une différence sur le nombre de dépendances, nous décidons d'attribuer le degré intermédiaire 2 au profit de

Stabilité	React	Vue	Svelte	Angular	Preact
Nombre de dépendances du framework	93	74	37	150	48
Changelog	Oui	Oui (dans la partie release)	Oui	Oui	Oui (dans la partie release)
Financement	Soutenu	Bénévolat	Bénévolat	Soutenu	Bénévolat
Outils d'aide à la mise à jour	Non	Non	Non	Oui (outils comparatif)	Page explicative
Année création	2013	2016	2016	2014	2015

FIGURE 5.12 – AHP : résultat analyse stabilité pour chaque framework

Stabilité	React	Vue	Svelte	Angular	Preact
React	1,00	7,00	6,00	2,00	4,00
Vue	0,14	1,00	0,50	0,17	0,33
Svelte	0,17	2,00	1,00	0,20	0,50
Angular	0,50	6,00	5,00	1,00	4,00
Preact	0,25	3,00	2,00	0,25	1,00
Total	2,06	19,00	14,50	3,62	9,83

FIGURE 5.13 – AHP : réciprocité pour le critère stabilité

Stabilité	React	Vue	Svelte	Angular	Preact	Total par ligne	Vecteur priorité (poids)
React	0,486	0,368	0,414	0,553	0,407	2,228	0,446
Vue	0,069	0,053	0,034	0,046	0,034	0,236	0,047
Svelte	0,081	0,105	0,069	0,055	0,051	0,361	0,072
Angular	0,243	0,316	0,345	0,276	0,407	1,587	0,317
Preact	0,121	0,158	0,138	0,069	0,102	0,588	0,118
Total	1,00	1,00	1,00	1,00	1,00	5,00	1,00

FIGURE 5.14 – AHP : normalisation et poids pour le critère stabilité

Svelte. En comparant Vue et Angular, on observe les mêmes différences que le framework React, afin d'être cohérent, nous attribuons le degré 6 au profit d'Angular. En comparant Vue et Preact, on observe plusieurs différences mineures, il n'y a qu'un an de différence entre leur création, Preact a moins de dépendances et fournit une aide à la mise à jour, pour ces raisons nous attribuons le degré 3 au profit de Preact. En comparant Svelte et Angular, nous avons les mêmes observations que pour la comparaison entre Angular et Vue, afin d'être cohérent, nous attribuons le degré 5 au profit d'Angular. En comparant Svelte et Preact, nous trouvons des différences mineures au profit de Preact, Svelte n'offre pas d'aide à la mise à jour, il a une longévité plus courte que Preact néanmoins il possède moins de dépendances que Preact, pour ces raisons nous attribuons le degré 2 au profit de Preact. En comparant Angular et Preact, nous observons que Angular a plus de dépendances mais fournit un outil d'aide à la mise à jour, il est soutenu financièrement et a une longévité plus longue, pour ces raisons, nous attribuons le degré 4 au profit d'Angular.

Ensuite nous normalisons les valeurs obtenues dans chaque cellule afin de calculer leur total et finalement leur poids (appelé aussi vecteur de priorité) voir figure 5.14.

Finalement nous calculons le CR (voir figure 5.15), ce qui nous donne une valeur de 0.025, 0.025 étant inférieur à 0.1, ceci indique que les poids définis sont acceptables et offrent un bon niveau de consistance.

Attractivité

La compilation des informations concernant ce critère est consultable via cette figure 5.16. Sur base de ce tableau nous construisons une table de réciprocité où nous comparons les frameworks les uns avec les autres en attribuant pour chaque pair un degré d'importance allant de 1 à 9 (voir figure 5.1), voir figure 5.17.

Stabilité	React	Vue	Svelte	Angular	Preact	Vecteur priorité (VP)	Nouveau vecteur priorité (VR)	VR/VP	RC	
React	1,00	7,00	6,00	2,00	4,00	0,446	2,315	5,197	CI	0,02781526
Vue	0,14	1,00	0,50	0,17	0,33	0,047	0,239	5,057		
Svelte	0,17	2,00	1,00	0,20	0,50	0,072	0,363	5,029	RI	1,11
Angular	0,50	6,00	5,00	1,00	4,00	0,317	1,656	5,217		
Preact	0,25	3,00	2,00	0,25	1,00	0,118	0,595	5,057		
							Somme	25,556	CR	0,0250588
							Lambda max	5,111		

FIGURE 5.15 – AHP : CR pour le critère stabilité

Attractivité	React	Vue	Svelte	Angular	Preact
Nombre étoiles sur github	169.000	183.000	47.100	73.300	29.100
Nombre de téléchargement	1.105.899.503	215.428.308	8.307.816	96.756.214	40.937.495
Nombre total de pages web référençant le framework	279.000.000	104.000.000	5.010.000	18.500.000	3.970.000
Satisfaction ranking	2	3	1	8	5

FIGURE 5.16 – AHP : résultat analyse attractivité pour chaque framework

En comparant React et Vue, nous remarquons une légère différence sur plusieurs points, nous décidons donc d'attribuer le degré 3 au profit de React. En comparant React et Svelte, on remarque que React a plus de trois fois plus d'étoiles sur Github, malgré que Svelte ait moins de téléchargements et moins de résultats retournés par Google, il est néanmoins le grand vainqueur de l'enquête de satisfaction, nous décidons donc d'attribuer le degré 5 au profit de React. En comparant React et Angular, nous voyons que React est supérieur sur tous les points, nous décidons donc d'attribuer le degré 7 au profit de React. En comparant React et Preact, tout comme pour Angular, nous observons que React est supérieur sur tous les points mais avec un écart plus prononcé, nous attribuons le degré 9 au profit de React afin d'être cohérent avec notre attribution précédente. En comparant Vue et Svelte, bien que Svelte soit le framework le plus satisfaisant Vue n'est pas loin derrière et obtient de meilleurs résultats sur toutes les autres métriques, nous décidons donc d'attribuer le degré 4 au profit de Vue. En comparant Vue et Angular, on observe que Vue obtient de meilleurs résultats sur tous les points, nous décidons d'attribuer le degré 6 au profit de Vue. En comparant Vue et Preact, nous arrivons à la même conclusion que pour Angular néanmoins Preact ayant le moins d'étoiles et Google renvoyant le moins de résultats pour ce framework nous prenons un degré un peu plus fort, nous décidons donc d'attribuer le degré 8 au profit de Vue. En comparant Svelte et Angular, nous remarquons que Angular a de meilleurs résultats que Svelte, néanmoins Svelte a la meilleure satisfaction, nous décidons donc d'attribuer le degré 4 au profit de Svelte. En comparant Svelte et Preact, nous remarquons que Svelte est supérieur sur tous les points excepté le nombre

Attractivité	React	Vue	Svelte	Angular	Preact
React	1,00	3,00	5,00	7,00	9,00
Vue	0,33	1,00	4,00	6,00	8,00
Svelte	0,20	0,25	1,00	4,00	5,00
Angular	0,14	0,17	0,25	1,00	3,00
Preact	0,11	0,13	0,20	0,33	1,00
Total	1,79	4,54	10,45	18,33	26,00

FIGURE 5.17 – AHP : réciprocité pour le critère attractivité

Attractivité	React	Vue	Svelte	Angular	Preact	Total par ligne	Vecteur priorité (poids)
React	0,560	0,661	0,478	0,382	0,346	2,426	0,485
Vue	0,187	0,220	0,383	0,327	0,308	1,424	0,285
Svelte	0,112	0,055	0,096	0,218	0,192	0,673	0,135
Angular	0,080	0,037	0,024	0,055	0,115	0,310	0,062
Preact	0,062	0,028	0,019	0,018	0,038	0,165	0,033
Total	1,00	1,00	1,00	1,00	1,00	5,00	1,00

FIGURE 5.18 – AHP : normalisation et poids pour critère attractivité

Attractivité	React	Vue	Svelte	Angular	Preact	Vecteur priorité (VP)	Nouveau vecteur priorité (VR)	VR/VP	RC	
React	1,00	3,00	5,00	7,00	9,00	0,485	2,746	5,658	CI	0,092164
Vue	0,33	1,00	4,00	6,00	8,00	0,285	1,622	5,695		
Svelte	0,20	0,25	1,00	4,00	5,00	0,135	0,717	5,324	RI	1,11
Angular	0,14	0,17	0,25	1,00	3,00	0,062	0,312	5,022		
Preact	0,11	0,13	0,20	0,33	1,00	0,033	0,170	5,144		
							Somme	26,843	CR	0,08303063
							Lambda max	5,369		

FIGURE 5.19 – AHP : CR pour critère attractivité

de téléchargement, nous décidons donc d'attribuer le degré 5 au profit de Svelte. En comparant Angular et Preact, nous remarquons qu'Angular est supérieur sur tous les points excepté pour la satisfaction où Angular obtient le résultat le plus bas, nous décidons donc d'attribuer le degré 3 au profit d'Angular.

Ensuite nous normalisons les valeurs obtenues dans chaque cellule afin de calculer leur total et finalement leur poids voir figure 5.18.

Finalement nous calculons le CR (voir figure 5.19), ce qui nous donne une valeur de 0.083, 0.083 étant inférieur à 0.1, ceci indique que les poids définis sont acceptables et offrent un bon niveau de consistance.

Performance

La compilation des informations concernant ce critère est consultable via cette figure 5.20. Sur base de ce tableau nous construisons une table de réciprocité où nous comparons les frameworks les uns avec les autres en attribuant pour chaque pair un degré d'importance allant de 1 à 9 (voir figure 5.1), voir figure 5.21.

En comparant React avec Vue, nous observons que la librairie de React est plus légère et le résultat du plugin Lighthouse est meilleur [18], nous avons décidé d'attribuer le degré 5 au profit de React. En comparant React et Svelte, on observe que Svelte est supérieur sur tous les points mais de peu, nous décidons donc d'attribuer le degré 3 au profit de Svelte. En comparant React et Angular, on observe que React est beaucoup plus légère et que la note attribuée est également supérieure, nous décidons d'attribuer le degré 7 au profit de React. En comparant React et Preact, on observe peu de différences, mais vu le score de Preact, nous décidons d'attribuer le degré 3 au profit de Preact. En comparant Vue et Svelte, Svelte surclasse Vue sur tous les points, nous décidons

Performance	React	Vue	Svelte	Angular	Preact
Temps chargement page todolist (ms)	307	309	307	304	309
Poids framework minifié + gzip (kb)	2,8	22,9	1,6	62,3	4
Note global outils Lighthouse	97	79	100	74	100

FIGURE 5.20 – AHP : résultat analyse performance pour chaque framework

Performance	React	Vue	Svelte	Angular	Preact
React	1,00	5,00	0,33	7,00	0,33
Vue	0,20	1,00	0,14	3,00	0,17
Svelte	3,00	7,00	1,00	9,00	3,00
Angular	0,14	0,33	0,11	1,00	0,17
Preact	3,00	6,00	0,33	6,00	1,00
Total	7,34	19,33	1,92	26,00	4,67

FIGURE 5.21 – AHP : réciprocité pour le critère performance

Performance	React	Vue	Svelte	Angular	Preact	Total par ligne	Vecteur priorité (poids)
React	0,136	0,259	0,174	0,269	0,071	0,909	0,182
Vue	0,027	0,052	0,074	0,115	0,036	0,304	0,061
Svelte	0,409	0,362	0,521	0,346	0,643	2,280	0,456
Angular	0,019	0,017	0,058	0,038	0,036	0,169	0,034
Preact	0,409	0,310	0,174	0,231	0,214	1,338	0,268
Total	1,00	1,00	1,00	1,00	1,00	5,00	1,00

FIGURE 5.22 – AHP : normalisation et poids pour le critère performance

donc d'attribuer le degré 7 au profit de Svelte. En comparant Vue et Angular, bien que le poids du framework Angular soit plus lourd on remarque une légère différence de résultat avec l'outil Lighthouse [18], nous décidons donc d'attribuer le degré 3 au profit de Vue. En comparant Vue et Preact, on remarque les mêmes différences qu'avec le framework Svelte, nous décidons d'attribuer le degré 6 au profit de Preact (celui-ci ayant des valeurs moins bonnes que Svelte). En comparant Svelte et Angular, on remarque que la différence est la plus extrême, Angular étant le plus lent et Svelte le plus rapide et plus léger, c'est la raison pour laquelle nous attribuons le degré 9 au profit de Svelte. En comparant Svelte et Preact, on remarque des différences négligeables au profit de Svelte, on décide donc d'attribuer le degré 3 au profit de Svelte. En comparant Preact et Angular, nous avons déjà observé que Preact avait des performances légèrement inférieure à Svelte, nous décidons donc d'attribuer le degré 6 au profit de Preact afin d'être cohérent avec nos attributions précédentes.

Ensuite nous normalisons les valeurs obtenues dans chaque cellule afin de calculer leur total et finalement leur poids voir figure 5.22.

Finalement nous calculons le CR (voir figure 5.23), ce qui nous donne une valeur de 0.083, 0.083 étant inférieur à 0.1, ceci indique que les poids définis sont acceptables et offrent un bon niveau de consistance.

Performance	React	Vue	Svelte	Angular	Preact	Vecteur priorité (VP)	Nouveau vecteur priorité (VR)	VR/VP	RC	
React	1,00	5,00	0,33	7,00	0,33	0,182	0,964	5,300	CI	0,09211727
Vue	0,20	1,00	0,14	3,00	0,17	0,061	0,308	5,062		
Svelte	3,00	7,00	1,00	9,00	3,00	0,456	2,534	5,556	RI	1,11
Angular	0,14	0,33	0,11	1,00	0,17	0,034	0,175	5,194		
Preact	3,00	6,00	0,33	6,00	1,00	0,268	1,533	5,730		
							Somme	26,842	CR	0,08298853
							Lambda max	5,368		

FIGURE 5.23 – AHP : CR pour le critère performance

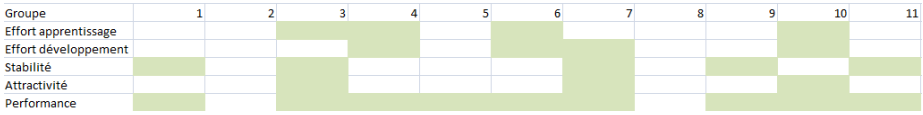

FIGURE 5.24 – Fréquence mention critères par articles

Objectif	L	D	S	A	P
L	1,00	1,00	0,33	3,00	0,14
D	1,00	1,00	0,33	3,00	0,14
S	3,00	3,00	1,00	4,00	0,20
A	0,33	0,33	0,25	1,00	0,13
P	7,00	7,00	5,00	8,00	1,00
Total	12,33	12,33	6,92	19,00	1,61

FIGURE 5.25 – AHP : Réciprocité des critères

5.5.4 Calcul poids critères

Cette étape consiste à définir le poids de chaque critère afin de pouvoir déterminer leur importance les uns par rapport aux autres. Désireux d'avoir une approche systématique, nous avons décidé de prendre en compte la fréquence de leurs mentions dans les articles repris de notre recherche de la littérature scientifique. Nous reprenons dans la figure suivante 5.24 le résultat de ce référencement.

Nous observons que l'effort de développement et d'apprentissage sont référencés quatre fois, la stabilité est référencée CI nq fois, l'attractivité trois fois et finalement la performance est référencée neuf fois. Nous pouvons déjà établir que la performance est le critère revenant le plus souvent ce qui en fait le critère le plus important.

Ensuite nous construisons une table de réciprocité où nous comparons les critères les uns avec les autres en attribuant pour chaque pair un degré d'importance allant de 1 à 9 (voir figure 5.1) en nous basant sur leur fréquence plus haut, voir figure 5.25. L'effort d'apprentissage est mentionné autant de fois que l'effort de développement, ce qui explique que la valeur 1 ait été attribuée dans les deux sens. L'effort d'apprentissage et l'effort de développement sont mentionnés une fois de plus que l'attractivité ce qui indique une importance supérieure modérée, nous avons donc attribué le degré 3 au profit de l'effort d'apprentissage et l'effort de développement. La stabilité est mentionnée une fois de plus que l'effort d'apprentissage et l'effort de développement, pour être cohérent avec notre choix précédent, nous avons attribué un degré 3 au profit de la stabilité. La performance étant citée dans 5 autres articles, nous avons identifié une très grande importance ce qui nous a menés vers le degré 7 au profit de la performance (nous n'avons pas utilisé le degré 9 car la performance n'est pas cités dans tous les articles). La stabilité étant citée dans deux articles supplémentaires par rapport à l'attractivité, l'importance est un peu plus forte que modérée mais n'est pas significative au point d'être essentielle, nous avons donc fait un compromis et sélectionné le degré 4 au profit de la stabilité. La stabilité

Objectif	L	D	S	A	P	Total par ligne	Vecteur priorité (poids)
L	0,081	0,081	0,048	0,158	0,089	0,457	0,091
D	0,081	0,081	0,048	0,158	0,089	0,457	0,091
S	0,243	0,243	0,145	0,211	0,124	0,966	0,193
A	0,027	0,027	0,036	0,053	0,078	0,220	0,044
P	0,568	0,568	0,723	0,421	0,621	2,900	0,580
Total	1,00	1,00	1,00	1,00	1,00	5,00	1,00

FIGURE 5.26 – AHP : Normalisation et poids des critères

Objectif	L	D	S	A	P	Vecteur priorité (VP)	Nouveau vecteur priorité (VR)	VR/VP	RC	
L	1,00	1,00	0,33	3,00	0,14	0,091	0,459	5,023	CI	0,0510893
D	1,00	1,00	0,33	3,00	0,14	0,091	0,459	5,023		
S	3,00	3,00	1,00	4,00	0,20	0,193	1,031	5,338	RI	1,11
A	0,33	0,33	0,25	1,00	0,13	0,044	0,228	5,172		
P	7,00	7,00	5,00	8,00	1,00	0,580	3,171	5,467		
							Somme	26,022	CR	0,0460264
							Lambda max	5,204		

FIGURE 5.27 – AHP : CR pour critères

étant moins citée que la performance mais étant néanmoins plus cité que l'effort de développement ou l'effort d'apprentissage, nous avons défini une importance forte de la performance, nous avons donc sélectionné le degré 5 au profit de la performance. Finalement l'attractivité étant le critère la moins cité, pour être cohérent avec l'effort de développement et l'effort d'apprentissage, nous avons sélectionné le degré 8 au profit de la performance.

Ensuite nous normalisons les valeurs obtenues dans chaque cellule afin de calculer leur total et finalement leur poids (appelé aussi vecteur de priorité) voir figure 5.26.

Finalement nous calculons le CR (voir figure 5.27), ce qui nous donne une valeur de 0.046, 0.046 étant inférieur à 0.1, ceci indique que les poids définis sont acceptables et offrent un bon niveau de consistance.

5.5.5 Classement des alternatives

Nous avons calculé le poids de chaque critère et nous avons calculé le poids de chaque alternative pour chacun de ces critères. Nous n'avons plus qu'à assembler toutes ces informations pour identifier le meilleur choix de framework parmi toutes les alternatives.

D'abord nous devons créer une première matrice 5x5 (alternative x critères), puis remplir dans chaque cellule le poids correspondant calculé précédemment et une seconde matrice 5x1 reprenant les poids des critères calculés précédemment. Ensuite il faut multiplier ces matrices et nous obtenons une matrice résultante contenant le poids proportionné de chaque alternative pour tous les critères, (voir figure 5.28).

Vous pouvez voir le résultat final avec cette figure 5.29.

5.6 Discussion résultats

Discussion général

Tout d'abord, nous avons différencié les poids des critères identifiés, nous observons que le critère le plus important est la performance et le critère le moins

	Poids des alternatives par critère					Poids des critères	Poids final des alternatives
	Effort apprentissage	Effort développement	Stabilité	Attractivité	Performance		
React	0,115	0,09	0,446	0,485	0,182	0,091	0,231633
Vue	0,152	0,144	0,047	0,285	0,061	0,091	0,083927
Svelte	0,388	0,485	0,072	0,135	0,456	0,193	0,363759
Angular	0,04	0,054	0,317	0,062	0,034	0,044	0,092183
Preact	0,306	0,227	0,118	0,033	0,268	0,58	0,228169

FIGURE 5.28 – AHP : Classement des alternatives

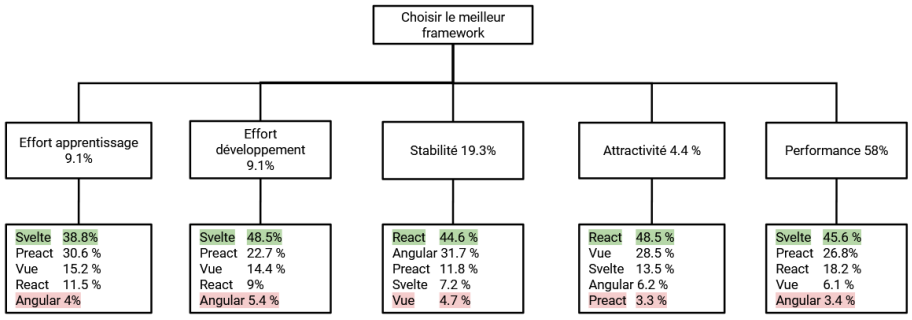

FIGURE 5.29 – AHP : Arbre complet

important est l'attractivité. Nous n'avons aucun élément spécifique à rajouter sauf qu'il est important de noter que ce résultat aurait pu être complètement différent si nous avions utilisé une méthode d'évaluation différente que celle basé sur la mention des critères dans les articles retenus de notre étude de la littérature scientifique.

Ensuite nous avons différencié les frameworks les uns aux autres pour chaque critère.

Concernant l'effort d'apprentissage nous observons que le meilleur framework est Svelte et le pire est Angular, en effet Svelte offre la barrière la plus fine d'apprentissage, il n'impose pas d'outils ni de langages intermédiaire et offre une documentation quasi complète tandis que Angular offre une courbe d'apprentissage plus lourde, il faut apprendre Typescript, il faut également installer NodeJs et utiliser leur CLI. Il est important de noter que Svelte n'a pas encore été mentionné dans aucun article repris de la littérature scientifique par contre Angular est référencé plusieurs fois, et la difficulté liée à l'apprentissage est mentionné à chaque fois [30, 31].

Concernant l'effort de développement nous observons que le meilleur framework est Svelte et le pire est Angular, l'une des forces d'Angular est qu'il offre un CLI facilitant grandement le travail du développeur, nous remarquons que Svelte n'offre pas de service équivalent pour la simple raison qu'il n'impose pas autant de dépendances et de couches intermédiaires, le CLI pour Svelte n'est pas nécessaire, finalement les métriques utilisées concernant l'application TodoList sont toutes en faveur de Svelte, il est très facile d'initialiser un projet, il est également très facile de démarrer son projet, nous remarquons également que développer l'application TodoList avec Svelte a été très rapide.

Concernant la stabilité nous observons que le meilleur framework est React et le pire est Vue, en effet React est le framework qui a la plus longue longévité et il est soutenu financièrement par Facebook tandis que Vue est l'un des frameworks

les plus jeunes (avec Svelte) et n'est pas soutenu financièrement par un grand groupe.
Concernant l'attractivité nous observons que le meilleur framework est React et le pire Preact, en effet le nombre d'articles disponible, le nombre de questions sur Stackoverflow, le nombre de téléchargements sont tous en faveur du framework React, Preact qui se positionne comme une alternative à React ne semble pas être assez connu.
Concernant la performance nous observons que le meilleur framework est Svelte et le pire est Angular, nous ne sommes pas étonné concernant Angular car il était déjà fait mention de problèmes de performances dans la littérature scientifique [31]. En ce qui concerne Svelte le résultat correspond aux objectifs définis par les créateurs du framework, leur ambition était de combler les problèmes des frameworks existant en améliorant le mécanisme de mise à jour du DOM en ne mettant à jour que les zones strictement nécessaires, ce résultat est confirmé par le résultat de l'outil Lighthouse [18].
Finalement le classement global désigne Svelte comme meilleur framework et Vue comme le pire framework (Angular est très proche également). Il est très difficile de comparer ce résultat global à l'état actuel de l'art sachant que la liste d'alternatives et les critères ne sont pas les mêmes, mais nous pouvons quand même constater que dans la littérature il existe 2 articles utilisant une partie des frameworks retenue chez nous qui arrivent à des conclusions que nous pouvons discuter. Dans leur article YongKang Xing, JiaPeng Huang et YongYao Lai [56] arrivaient à la conclusion que dans le cadre du développement d'un e-commerce il est fortement conseillé de choisir le framework Angular 2 alors que React et Vue sont plus adaptés pour des applications plus petites ou moyennes (ex : blog, communication, ...). Si nous comparons nos résultats sur base de la stabilité, nous remarquons que nous n'arrivons pas au même résultat, React est supérieur à Angular. Si nous comparons sur base de la performance nous voyons que Angular est le pire framework a sélectionné, nous doutons qu'un framework ayant des problèmes de performances soit le plus adéquat dans le développement d'une application e-commerce. Dans leur article Ankush Ravindra Gochke et Mr. Prakash Ken [11] arrivaient à la conclusion que React était le meilleur framework à utiliser quel que soit le contexte. Il est intéressant de noter que du point de vue de la stabilité et de l'attractivité nous arrivons à la même conclusion néanmoins pour tous les autres critères pris individuellement React ne se distingue pas particulièrement. Mais il reste important de mentionner que React est le second meilleur framework à sélectionner pour tous les critères.

Limites

Bien que la méthode AHP soit très utile pour sélectionner des alternatives sur base d'une liste de critères, nous avons rencontré quelques limitations. Cette méthode manque de flexibilité, une fois que tous les poids ont été calculé pour différencier les critères entre eux, il n'est pas possible de faire une composition limitée de la liste complète des critères sans recalculer les nouveaux poids. Par exemple : dans notre cas, il n'est pas possible de prendre un sous-ensemble de critères comme "Attractivité,Performance" et de récupérer le meilleur framework pour ces 2 critères sans devoir recalculer leurs poids respectifs. Cette contrainte existe également pour les alternatives, vouloir rajouter ou enlever un framework affecterait les résultats et impliquerait de refaire les calculs de poids.

L'attribution des degrés d'importance (entre 1 et 9) est une tâche fastidieuse car elle nécessite une argumentation avec des données testables et le degré choisi pourrait être subjectif, par exemple il peut être difficile de faire la différence entre le degré 5 et le degré intermédiaire 6.

Chapitre 6

Conclusion et perspectives

Nous avons consacré notre étude sur la comparaison qualitative et quantitative des frameworks Javascript dans le cadre du développement web côté frontend qui nous a mené à identifier plusieurs questions de recherche :

- QR1 : Existe t'il une méthode efficace de comparaison et sélection d'un framework Javascript dans le cadre du développement d'une application web ?
- QR2 : Quels sont les critères retenus pour comparer des framework Javascript ?
- QR3 : Est-ce que tous les frameworks Javascript sont pris en compte ?

Pour la question de recherche QR1, nous avons trouvé plusieurs comparaisons dans la littérature scientifique, mais il semble qu'il n'existe pas de standard ou de comparaison unanimement reconnue, correct et intemporel.
Pour la question de recherche QR2, notre recherche nous a permis d'identifier 4 groupes de critères :

- Les critères mesurables
- Les fonctionnalités désirées/attendues
- Le résultat de l'exécution de logiciels externes
- Les critères statiques ou mesurables mais peu fiables car changeant

Finalement pour la question de recherche QR3, nous n'avons pas trouvé de solution intemporelle, de nouveaux frameworks sont créé tous les ans, les frameworks existants sont améliorés et parfois refactoré sans offrir de rétrocompatibilité (ex : Angular v1 vs Angular v2) et finalement il peut arriver que des frameworks n'assurent plus de support.
Ce constat implique quelle que soit la solution mise en place d'un coût de maintenabilité afin de gérer les nouvelles adaptations (création, modification, nouvelle version, suppression).

Face à ce constat, nous sommes arrivé à 2 conclusions :

- Il n'existe pas de réponse catégorique fiable et unanimement acceptée sur le meilleur framework Javascript à utiliser pour le développement d'une application web côté frontend
- Nous n'avons pas trouvé de comparaison utilisant les arbres de décision dans la littérature scientifique

Nous avons donc proposé une solution amenant à un résultat sous forme d'arbre de décision, nous avons utilisé la méthode AHP qui est indiquée pour ce type de problème. Avant de construire notre arbre nous avons dû choisir une popu-

lation cible de frameworks ainsi qu'un ensemble de critères, notre volonté étant de fournir une étude reproduisible, nous avons détaillé toutes les étapes de la méthode. Notre étude nous a permit d'obtenir six réponses :

— Effort apprentissage : Svelte
— Effort développement : Svelte
— Stabilité : React
— Attractivité : React
— Performance : Svelte
— Tous les critères : Svelte

Nous avons pu comparer nos résultats à 2 articles issus de notre étude de la littérature scientifique, nous avons trouvé des points de divergences qui étaient à chaque fois en la défaveur de l'article provenant de la littérature scientifique mais également des résultats confirmant leur conclusion. Globalement nous pensons que l'arbre résultant offre la meilleure solution à cette problématique.

Néanmoins l'application de la méthode AHP implique deux contraintes, un manque de flexibilité et une subjectivité potentielle dans l'attribution des degrés d'importance. Il serait intéressant de pouvoir sélectionner une partie des critères et/ou des alternatives pour voir l'influence que ça occasionne sur le résultat final. Madame Veena Nayak à recenser plusieurs méthodes dans son étude qui pourrait être un point de départ intéressant pour une étude plus approfondie [5].

Bibliographie

[1] *Angular*, dernière visite : 14/04/2021. `https://angular.io/`.

[2] T. J. Berners-Lee, R. Cailliau, and J.-F. Groff. The world-wide web. *Comput. Netw. ISDN Syst.*, 25(4–5) :454–459, November 1992.

[3] Tim Berners-Lee, Robert Cailliau, Jean-François Groff, and Bernd Pollermann. World-wide web : the information universe. *Internet Res.*, 20 :461–471, 2010.

[4] Jim Conallen. Modeling web application architectures with uml. *Commun. ACM*, 42(10) :63–70, October 1999.

[5] Rio D'souza and Veena Nayak. A survey on multi-criteria decision making methods in software engineering, 07 2018.

[6] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, 2(2) :115–150, May 2002.

[7] Mozilla Firefox. Accessibility, 2021. URL : `https://developer.mozilla.org/en-US/docs/Web/Accessibility`. dernière visite : 15/05/2021.

[8] Hans Gellersen and Martin Gaedke. Object-oriented web application development. *Internet Computing, IEEE*, 3 :60 – 68, 02 1999.

[9] *Github : Polymer*, dernière visite : 12/04/2021. `https://github.com/Polymer/polymer`.

[10] Andreas Gizas, Sotiris Christodoulou, and Theodore Papatheodorou. Comparative Evaluation of Javascript Frameworks. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12 Companion, pages 513–514, New York, NY, USA, 2012. Association for Computing Machinery. event-place : Lyon, France.

[11] Ankush Gochke. Single Page Web Application Technologies. *International Journal for Research in Applied Science and Engineering Technology*, 7 :1692–1697, 2019.

[12] Google. First Contentful Paint, 2019. URL : `https://web.dev/first-contentful-paint/`. dernière visite : 17/05/2021.

[13] Google. First CPU Idle, 2019. URL : `https://web.dev/first-cpu-idle/`. dernière visite : 17/05/2021.

[14] Google. First Meaningful Paint, 2019. URL : `https://web.dev/first-meaningful-paint/`. dernière visite : 17/05/2021.

[15] Google. Max Potential First Input Delay, 2019. URL : `https://web.dev/lighthouse-max-potential-fid/`. dernière visite : 17/05/2021.

[16] Google. Speed Index, 2019. URL : https://web.dev/speed-index/. dernière visite : 17/05/2021.

[17] Google. Time To Interactive, 2019. URL : https://web.dev/interactive/. dernière visite : 17/05/2021.

[18] Google. Lighthouse, 2021. URL : https://developers.google.com/web/tools/lighthouse. dernière visite : 16/05/2021.

[19] Daniel Graziotin and Pekka Abrahamsson. Making Sense out of a Jungle of JavaScript Frameworks : towards a Practitioner-friendly Comparative Analysis. 7983, 2013.

[20] Gaëlle Guesdon. Aide multicritère à la décision-Comparaison de Saaty, 2011. URL : https://www.gci.ulaval.ca/fileadmin/gci/documents/rgalvez/Cours%20en%20classe/power%20point%20_%20Guesdon/Cours%205e_Outils%20m%C3%A9thode%20de%20comparaison%20de%20Saaty.pdf. dernière visite : 05/04/2021.

[21] Holger Kienle. It's about Time to Take JavaScript (More) Seriously. *Software, IEEE*, 27 :60 – 62, 2010.

[22] Michael Xieyang Liu, Jane Hsieh, Nathan Hahn, Angelina Zhou, Emily Deng, Shaun Burley, Cynthia Taylor, Aniket Kittur, and Brad A. Myers. Unakite : Scaffolding Developers' Decision-Making Using the Web. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, UIST '19, pages 67–80, New York, NY, USA, 2019. Association for Computing Machinery. event-place : New Orleans, LA, USA.

[23] Sanjay Misra and Ferid Cafer. Estimating Quality of JavaScript. *International Arab Journal of Information Technology*, 9 :535–543, 2012.

[24] N. Nakajima, S. Matsumoto, and S. Kusumoto. Jact : A Playground Tool for Comparison of JavaScript Frameworks. In *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, pages 474–481, December 2019. ISSN : 2640-0715.

[25] NodeJs. *NodeJs*, dernière visite : 25/04/2021. https://nodejs.org/en/.

[26] Npm. *Npm*, dernière visite : 25/04/2021. https://www.npmjs.com/.

[27] *Pros and Cons of ReactJS Web App Development*, dernière visite : 12/04/2021. https://ddi-dev.com/blog/programming/pros-and-cons-reactjs-web-app-development/.

[28] Amantia Pano, Daniel Graziotin, and Pekka Abrahamsson. Factors and actors leading to the adoption of a JavaScript framework. *Empirical Software Engineering*, 2018.

[29] J. Raigoza and R. Thakkar. Browser Performance of JavaScript Framework, SAPUI5 jQuery. In *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1420–1421, December 2016.

[30] Miguel Ramos, Marco Valente, and Ricardo Terra. AngularJS Performance : A Survey Study. *IEEE Software*, 1 :1–11, 2017.

[31] Miguel Ramos, Marco Tulio Valente, Ricardo Terra, and Gustavo Santos. AngularJS in the Wild : A Survey with 460 Developers. In *Proceedings of the 7th International Workshop on Evaluation and Usability of Programming Languages and Tools*, PLATEAU 2016, pages 9–16, New York, NY, USA,

2016. Association for Computing Machinery. event-place : Amsterdam, Netherlands.

[32] Thomas L. Saaty. *Analytic Hierarchy Process*. American Cancer Society, 2005.

[33] Leon Shklar and Richard Rosen. Web Application Architecture : Principles, Protocols and Practices, 2003.

[34] Angular Update Guide. URL : `https://update.angular.io/`. dernière visite : 18/05/2021.

[35] rowser Market Share Worldwide, 2020. URL : `https://gs.statcounter.com/browser-market-share`. dernière visite : 16/05/2021.

[36] Analyse Syntaxique. URL : `https://developer.mozilla.org/en-US/docs/Glossary/Parse`. dernière visite : 25/05/2021.

[37] *Preact*, dernière visite : 12/04/2021. `https://preactjs.com/`.

[38] Upgrade guide Preact. URL : `https://preactjs.com/guide/v10/upgrade-guide/`. dernière visite : 18/05/2021.

[39] *ReactJs*, dernière visite : 12/04/2021. `https://reactjs.org/`.

[40] *Should you use Svelte in production ?*, dernière visite : 13/04/2021. `https://blog.logrocket.com/should-you-use-svelte-in-production/`.

[41] *Similar tech*, dernière visite : 28/04/2021. `https://www.similartech.com/`.

[42] *Svelte*, dernière visite : 13/04/2021. `https://svelte.dev/`.

[43] *The Good and the Bad of Vue.js Framework Programming*, dernière visite : 14/04/2021. `https://www.altexsoft.com/blog/engineering/pros-and-cons-of-vue-js/`.

[44] *Using Preact as a React Alternative*, dernière visite : 12/04/2021. `https://www.sitepoint.com/using-preact-react-alternative/`.

[45] *Vue*, dernière visite : 14/04/2021. `https://vuejs.org/`.

[46] M.N. Sudin, M. Salim, Mohd Rizal Alkahari, Mohd Abdullah, Muhd Ridzuan Mansor, Mohd Zaid Akop, and Faiz Ramli. Ahp method and application example for the robust multi-criteria design concept selection. *ARPN journal of engineering and applied science*, 12, 04 2017.

[47] *TodoMvc*, dernière visite : 25/04/2021. `https://todomvc.com/`.

[48] *Performance Testing Guidance for Web Applications. Microsoft Patterns & Practices.*, dernière visite : 12/08/2020. `https://docs.microsoft.com/en-us/previous-versions/msp-n-p/bb924375(v=pandp.10)?redirectedfrom=MSDN`.

[49] *Zotero : Your personal research assistant*, dernière visite : 09/08/2020. `https://www.zotero.org/`.

[50] W3Schools. *What is the HTML DOM ?*, dernière visite : 16/08/2020. `https://www.w3schools.com/whatis/whatis_htmldom.asp`.

[51] *What does react.js try to solve ? Can you provide a practical example ?*, dernière visite : 12/04/2021. `https://www.quora.com/What-does-react-js-try-to-solve-Can-you-provide-a-practical-example`.

[52] Wikipedia. *Front end and back end*, dernière visite : 16/08/2020. `https://en.wikipedia.org/wiki/Front_end_and_back_end`.

[53] Wikipedia. *Polyfill*, dernière visite : 16/08/2020. https://fr.wikipedia.org/wiki/Polyfill.

[54] *Wikipedia : ReactJs*, dernière visite : 12/04/2021. https://en.wikipedia.org/wiki/React_(JavaScript_library).

[55] Allen Wirfs-Brock and Brendan Eich. JavaScript : The First 20 Years. *Proc. ACM Program. Lang.*, 4(HOPL), June 2020. Place : New York, NY, USA Publisher : Association for Computing Machinery.

[56] YongKang Xing, JiaPeng Huang, and YongYao Lai. Research and Analysis of the Front-End Frameworks and Libraries in E-Business Development. In *Proceedings of the 2019 11th International Conference on Computer and Automation Engineering*, ICCAE 2019, pages 68–72, New York, NY, USA, 2019. Association for Computing Machinery. event-place : Perth, WN, Australia.

[57] *Unakite : Scaffolding Developers' Decision-Making Using the Web.*, dernière visite : 12/08/2020. https://www.youtube.com/watch?v=UMQ-kWgmbQ4.

Table des figures

Liste des tableaux

Glossaire

backend est issu de la séparation des responsabilités entre la couche de présentation et la couche d'accès aux données. Dans le cadre d'une application client-serveur, est considéré comme backend la partie du logiciel qui doit être exécuté par le serveur [52]. 31

DOM c'est un modèle objet en HTML, il est définis par les éléments HTML comme objets et des propriétés/méthodes/évènements pour tous les éléments HTML [50]. 11, 13, 17, 19

frontend est issu de la séparation des responsabilités entre la couche de présentation et la couche d'accès aux données. Dans le cadre d'une application client-serveur, est considéré comme frontend la partie du logiciel qui doit être exécuté par le client [52]. 3, 9, 12, 20, 23, 32, 49

parser "Parser" signifie analyser et convertir un programme en un format interne que l'environnement d'exécution peut exécuter, par exemple le moteur JavaScript dans les navigateurs.[36]. 7, 30

polyfill aussi nommé shim, ou encore prothèse d'émulation désigne un palliatif logiciel implémentant une rétrocompatibilité d'une fonctionnalité ajoutée à une interface de programmation dans des versions antérieures de cette interface [53]. 17

TodoList modèle typique d'application web permettant de gérer une liste de tâche [19, 29, 24]. 12, 13, 23, 31, 33, 35, 37, 38, 46

Acronymes

AHP Analyse hiérarchique des procédés. 1, 4, 23, 36–46, 49, 50, 55, 56
ajax Asynchronous JavaScript and XML. 11, 17, 19
API interface de programmation d'application. 8

CI L'index de consistance. 25, 44
CLI Interface de ligne de commande. 33, 38
CR Ratio de consistance. 25, 37–43, 45, 55, 56

ECMA European Computer Manufacturers Association. 8

HTML HyperText Markup Language. 5–7, 19
HTTP HyperText Transfer Protocol. 5

IETF Internet Engineering Task Force. 8

JCCM Javascript Cognitive Complexity Measure. 11, 18
JSX Javascript Xml. 19

POJO Plain old Javascript object. 13, 19

RI L'index randomisé. 25

SGML Standard Generalized Markup Language. 5

URL Univeral Resource Locator. 5

W3C W3 Consortium. 8
web World Wide Web. 3–9, 20, 23, 49, 55