Analyse interface graphique GED

But:

Remplacer le système de vue en arbre actuel en gardant l'ensemble des fonctionnalités tout en modernisant l'interface utilisateur.

Sujet :

Analyse de remplacement de l’interface graphique en arbre avec intégration d’Office 365.

Application :

Pour les utilisateurs Medhive incluant le module O365.

Table des matières

[Analyse interface graphique GED 1](#_Toc30067592)

[Table des matières 3](#_Toc30067593)

[1 Introduction 5](#_Toc30067594)

[1.1 Contexte 5](#_Toc30067595)

[1.1.1 ExtJS 5](#_Toc30067596)

[1.2 Fonctions requises 6](#_Toc30067597)

[2 Alternatives 6](#_Toc30067598)

[2.1 Plugin JavaScript 6](#_Toc30067599)

[2.1.1 Avantages / Inconvénients 6](#_Toc30067600)

[2.1.2 Propositions 6](#_Toc30067601)

[2.1.2.1 jsTree 6](#_Toc30067602)

[2.1.2.2 FancyTree 7](#_Toc30067603)

[2.1.3 Autres plugins 8](#_Toc30067604)

[2.2 Framework JavaScript 8](#_Toc30067605)

[2.2.1 Avantages / Inconvénients 8](#_Toc30067606)

[2.2.2 State of the Art 2019 9](#_Toc30067607)

[2.2.3 Propositions 9](#_Toc30067608)

[2.2.3.1 React.js 9](#_Toc30067609)

[2.2.3.2 Vue.js 10](#_Toc30067610)

[2.2.3.3 Svelte 11](#_Toc30067611)

[2.2.4 React vs Vue 11](#_Toc30067612)

[2.2.4.1 JSX vs Templates 12](#_Toc30067613)

[2.2.4.2 Portage mobile 12](#_Toc30067614)

[2.2.4.3 Développements 12](#_Toc30067615)

[2.2.4.4 Conclusion 13](#_Toc30067616)

[2.2.5 Autres frameworks 13](#_Toc30067617)

[2.3 MS OneDrive 13](#_Toc30067618)

[2.3.1 Avantages / Inconvénients 14](#_Toc30067619)

[Liens utiles 15](#_Toc30067620)

[Abréviations 15](#_Toc30067621)

[Historique 15](#_Toc30067622)

# Introduction

Ce document analyse différentes alternatives de remplacement de la vue en arbre intégré à MedHive via le plugin JavaSctipt ExtJS.

Le but principal est de remplacer le système actuel en gardant l'ensemble des fonctionnalités tout en modernisant l'interface utilisateur.

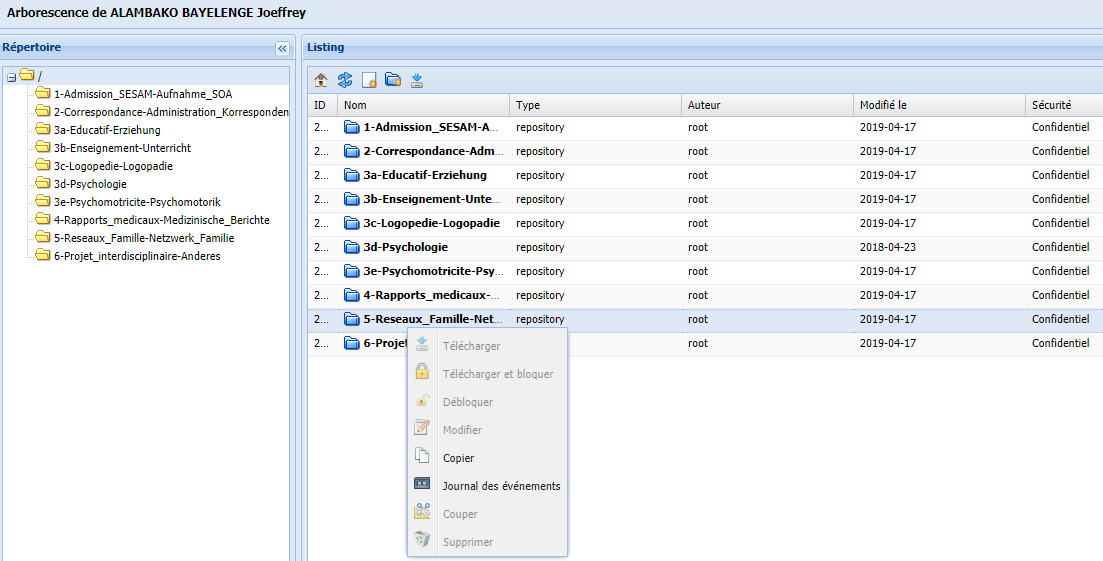
## Contexte

### ****ExtJS****

**ExtJS** est une bibliothèque JavaScript permettant de construire des applications web interactives. Ext apporte un certain nombre de composants visuels comme des champs de formulaires avancés, des arbres, des tableaux, menu et barre d'outils, onglets, boîtes de dialogue.

La version 2.0 est sortie le 4 décembre 2007. Il s'agit d'une amélioration majeure de la bibliothèque.

Dans OIH3, ExtJS est installé dans le module "download" sous javascript > extjs, l'interface proposée est la suivante.



## Fonctions requises

* Drag-and-drop de fichiers depuis l'ordinateur local vers OIH
* Drag-and-drop entre répertoires
* Multilinguisme (FR, EN, DE, ...)
* Meilleurs interface utilisateur
* Sélection multiple de fichiers / répertoires
* Vu en arbre applicable à plusieurs modules (documents, projets, activités, ...)

# Alternatives

Il y a 3 alternatives différentes pouvant être envisagées. Chacune présentant des avantages et des inconvénients ainsi qu’une charge de travail différente.

1. Plugin JavaScript
2. Framework JavaScript
3. MS OneDrive

## Plugin JavaScript

La première alternative consiste à utiliser un plugin JS ou jQuery existant et open-source. Une fois importé, le but est de le rendre « MedHive » compatible et de le customiser selon nos besoins.

C’est la solution qui s’apparente la plus à l’implémentation actuel utilisant ExtJS.

### Avantages / Inconvénients

Charge de travail estimé : 2/3

|  |  |
| --- | --- |
| **Avantages** | **Inconvénients** |
| Utilisation d’une technologie plus récente  Plusieurs fonctions déjà réalisée | Plus-value faible par rapport à ExtJS  Customisation limitée  Peu de support et de mise à jour |

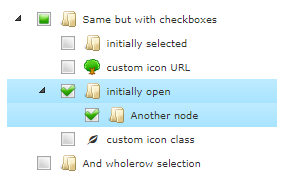
### Propositions

#### jsTree

jsTree est un plugin jQuery, qui fournit des arbres interactifs. Il est facilement extensible, thématique et configurable, il prend en charge les sources de données HTML et JSON et le chargement AJAX.

jsTree utilise le système d'événements de jQuery, donc la liaison de rappels sur divers événements de l'arborescence est familière et facile. Fonctions prises en charges : drag & drop, navigation au clavier, types de nœuds personnalisables…

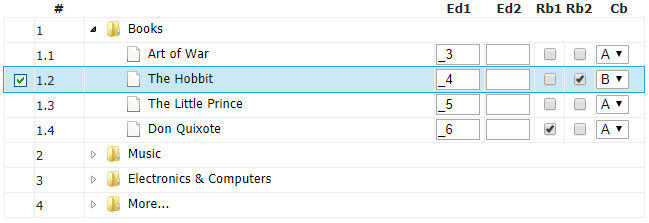
JsTree dispose d'une grande bibliothèque de plugins permettant d'étendre facilement ses fonctionnalités.



Site web & documentation : <https://www.jstree.com/>

#### FancyTree

Fancytree est un plugin JavaScript de vue d'arborescence avec prise en charge du clavier, de l'édition en ligne, du filtrage, des cases à cocher, du drag'n'drop et du lazy loading.



Site web & documentation : <https://github.com/mar10/fancytree>

### Autres plugins

La plupart de ces plugins partagent les mêmes fonctionnalités et sont customisable. On peut encore citer les plug-ins suivants :

* jqTree (<https://mbraak.github.io/jqTree/>)
* Bootstrap Treeview (<https://mdbootstrap.com/plugins/jquery/treeview/>)
* jQuery file tree (<http://jqueryfiletree.github.io/>)

## Framework JavaScript

La seconde alternative consiste à intégrer un Framework JS et à y adjoindre une librairie permettant de mettre en place une vue en arbre. C’est la solution la plus customisable et la plus polyvalente. Cette solution demande une phase d’apprentissage du Framework choisit.

Tous les frameworks JS présentés ici sont open-source.

### Avantages / Inconvénients

Charge de travail estimé : 3/3

|  |  |
| --- | --- |
| **Avantages** | **Inconvénients** |
| Customisable, polyvalent, performant  Vue réutilisable sur d’autres modules  Utilisation de standards  Connaissance acquises réutilisable ailleurs dans MedHive (formulaires)  Updates & communauté grandissante | Mise en place « lourde »  Phase d’apprentissage  Développement ou mise en place de chacune des fonctionnalités |

### State of the Art 2019

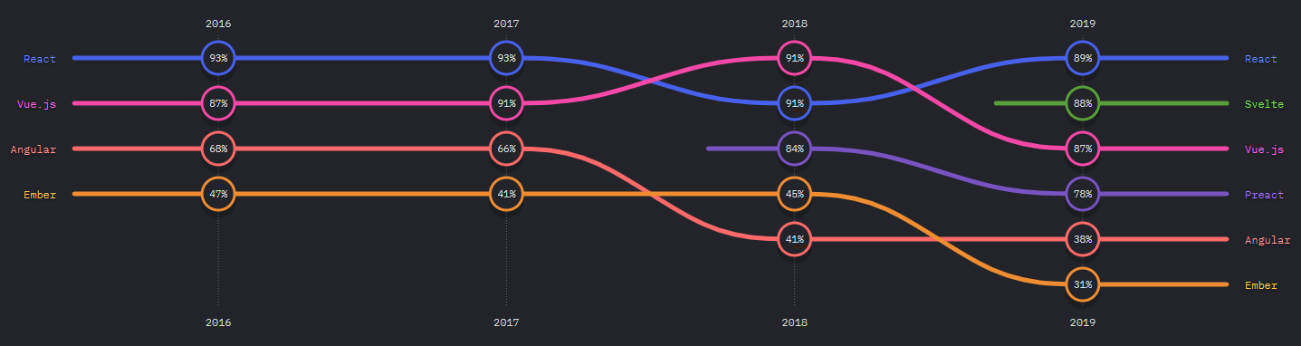


Figure 1 : <https://2019.stateofjs.com/front-end-frameworks/>

### Propositions

#### React.js

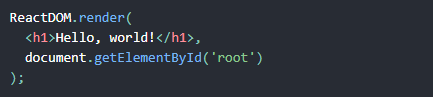
2013, Facebook, MVC, DOM virtuel, Flexible, Performant

React.js est une bibliothèque JavaScript libre développée par Facebook depuis 2013. React est une bibliothèque qui gère l'interface de l'application, considéré comme la vue dans le modèle MVC.

La bibliothèque se démarque de ses concurrents par sa flexibilité et ses performances, en travaillant avec un DOM virtuel et en ne mettant à jour le rendu dans le navigateur qu'en cas de nécessité.

Site web & documentation : <https://fr.reactjs.org/>

##### Hello, World !



##### Plugin

En complément du framework React, il y a la possibilité d’utiliser un des plugins suivant pour faciliter l’implémentation de la vue en arbre :

* deni-react-treeview (<https://reactjsexample.com/a-themable-and-configurable-treeview-for-react/> )
* react-treebeard (<https://github.com/storybookjs/react-treebeard>)

#### Vue.js

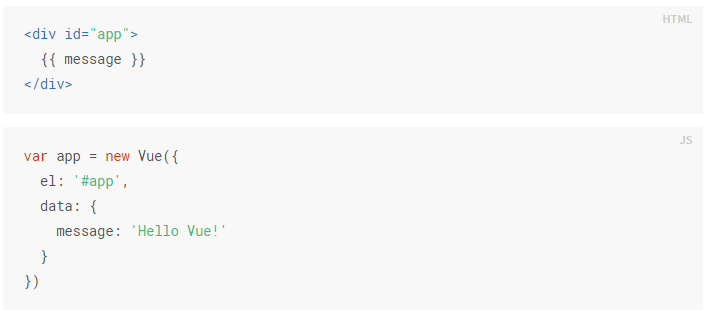
2014, Evan You (Google), Progressif, Déclaratif

Vue.js est un framework JavaScript open source pour la construction d'interfaces utilisateur et d'applications.

Contrairement à d'autres frameworks, Vue.js est conçu de A à Z pour être intégré progressivement. La bibliothèque principale se concentre uniquement sur la couche de "vue" et est facile à intégrer avec d'autres bibliothèques ou avec des projets existants.

Site web & documentation : <https://vuejs.org/>

##### Hello, World !



##### Plugin

En complément du framework React, il y a la possibilité d’utiliser un des plugins suivant pour faciliter l’implémentation de la vue en arbre :

* vue-treeview (<https://github.com/ll931217/vue-treeview>)
* vuejs-tree (<https://github.com/scalia/vuejs-tree>)

#### Svelte

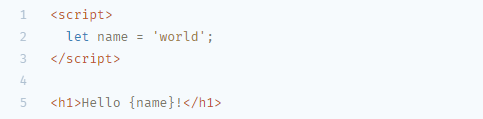
2016, Rich Harris, Compilation

Svelte est un framework JavaScript qui apporte une nouvelle approche pour créer des interfaces utilisateur. Contrairement aux frameworks traditionnels comme React et Vue qui effectuent la majeure partie de leur travail dans le navigateur, Svelte décale ce travail en une étape de compilation qui se produit lors de la création de l'application.

Au lieu d'utiliser des techniques telles que la différence de DOM virtuel, Svelte écrit du code qui met à jour chirurgicalement le DOM lorsque l'état de votre application change.

Site web & documentation : <https://svelte.dev/>

##### Hello, World !



### React vs Vue

La présentation des deux frameworks étant faite plus haut, on peut en conclure qu’ils sont plutôt similaires. En effet, Vue.js et React.js ont beaucoup d’éléments en commun notamment :

* L'architecture basée sur les composants
* L'utilisation du DOM virtuel
* L'utilisation d'accessoires
* Les outils de développement Chrome pour le débogage
* …

Les différences majeures notables entre ces deux frameworks sont expliquées dans les points ci-dessous.

#### JSX vs Templates

*React => JSX | Vue.js => templates (+ support JSX)*

Avec **React.js**, tous les composants expriment leur UI à travers des fonctions JSX (une syntaxe déclarative comme XML qui fonctionne au sein de JavaScript).

Les fonctions de rendu de JSX ont certains avantages :

* Utilisation de la puissance de JavaScript pour créer les vues. Cela inclut les variables temporaires, le contrôle de flux et les valeurs JavaScript directement référencées dans la portée.
* Les outils d’aide (analyse des erreurs, vérifications de typage, autocomplétions) sont plus avancés que ceux des templates de Vue.

Dans **Vue.js**, il existe aussi des fonctions de rendu et un support de JSX permettant de bénéficier de sa puissance. Toutefois Vue fonctionne avec des templates HTML permettant une meilleure expérience de développement

* Les templates HTML sont plus simples à migrer progressivement depuis une application existante.
* Pour les développeurs, les templates sont plus simples et plus naturels à écrire.

#### Portage mobile

Une différence cruciale entre les deux frameworks est le portage mobile du code.

Avec React, le code peut être partagé entre React.js et React Native, ce qui permet de porter une version mobile des interfaces.

Dans Vue, il existe un plugin (NativeScript) pour créer de vraies applications mobiles natives en utilisant Vue.js.

#### Développements

Pour ce qui est du développement en soit, on trouve autant d’avis que de développeurs, ceux-ci s’accordent sur les points suivants.

|  |  |
| --- | --- |
| **What devs love about React.js** | **What devs love about Vue.js** |
| Elegant programming style and patterns  Rich package ecosystem  Widespread usage | Easy learning curve  Elegant programming style and patterns  Good documentation |

#### Conclusion

On trouve énormément de comparatifs sur internet entre ces deux frameworks principaux. Ces différents comparatifs évoluent rapidement néanmoins ils peuvent être résumé par la citation suivante.

*« Picking React vs. Vue, totally depends on your business use-cases. If you are looking to build complex applications, with support React maybe a good choice. But if you are looking to build something very simple, easy with a fast turnaround time, you could go for Vue with the support of templates which are easy to learn and code. »*

### Autres frameworks

Il existe énormément de frameworks JS à l’heure actuelle, toutefois on peut encore citer les frameworks suivants pouvant être intéressants.

* Preact (<https://preactjs.com/>)
* Angular (<https://angularjs.org/>)
* Ember (<https://emberjs.com/>)

Autre plugin de vue en arbres pour framework JS :

* Bosket (<https://github.com/elbywan/bosket>)

## ~~MS OneDrive~~

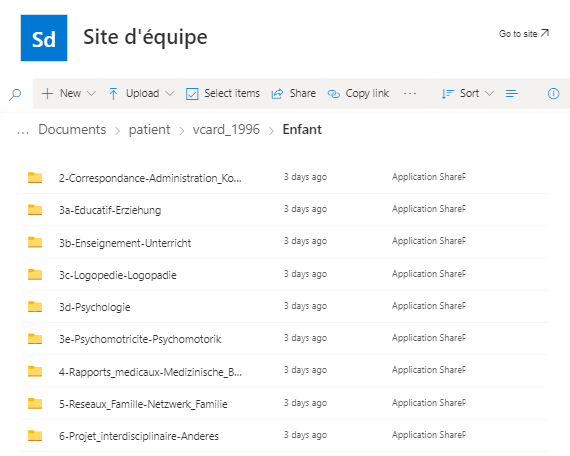
*EDIT suite à la séance du 20.01.2020 : Cette solution n’est pas envisageable.*

La dernière solution consiste à utiliser directement l’interface proposé par Office 365 pour la gestion des documents. La proposition est d’embeder l'interface de OneDrive directement dans MedHive et d'en tirer tous les avantages.

L’application d’un thème customisable est disponible sur « OneDrive for Business ». Cela nous permettrait d’adapter les couleurs à celles du client.

Sur cette solution, le rapport charge de travail / résultat est très bon. En effet la mise en place et simple et l’interface obtenue est très satisfaisante.

Site web & documentation : <https://docs.microsoft.com/en-us/sharepoint/dev/solution-guidance/customize-onedrive-for-business-sites>



### Avantages / Inconvénients

Charge de travail estimé : 1/3

|  |  |
| --- | --- |
| **Avantages** | **Inconvénients** |
| Responsive  Interface simple et intuitive  Multilinguisme supporté d’office  Toutes les fonctions d’une GED déjà implémentées | Non customisable  Pas réutilisable sur d’autres modules  Dépendant de Microsoft |

# Séance du 20.01.2020

## PV

Durant cette séance, ce document a été présenté et discuté. Il en est sorti les points suivants :

* Cette nouvelle interface doit être utilisable autant par les clients utilisant la GED O365 que par les clients utilisant la GED MedHive. De facto la solution 2.3 présenté plus haut n’est pas utilisable.
* La solution choisit doit être utilisable ailleurs dans OIH3 et doit y apporter une plus-value
* La question de garder ou non une navigation avec un arbre reste ouverte

Deux solutions alternatives ont alors été proposées, ceux deux solutions sont présentées ci-dessous.

## Nouvelles propositions de solutions

### Framework JS (Analyse approfondie)

Nous arrivons donc sur la conclusion qu’un analyse plus poussées permettant d’affirmer ou d’infirmer l’utilisation d’un Framework JS est nécessaire.

L’analyse du Framework JS doit :

* Présenter clairement les avantages et la plus-value obtenus
* Exposer différents composants retenus permettant de répondre aux besoins définis pour le GUI de la GED
* Assurer que le framework choisit soit le bon et correspondent au besoin futurs d’OIH3 (formulaires, tables…)
* Assurer une compatibilité avec différents OS / Navigateurs
* Minimiser la baisse de performance s’il y en a une

### Adaptation des tables actuels

La seconde solution proposée serait d’adapter les tables actuellement utilisées dans les activités pour y adjoindre un arbre et les services nécessaires.

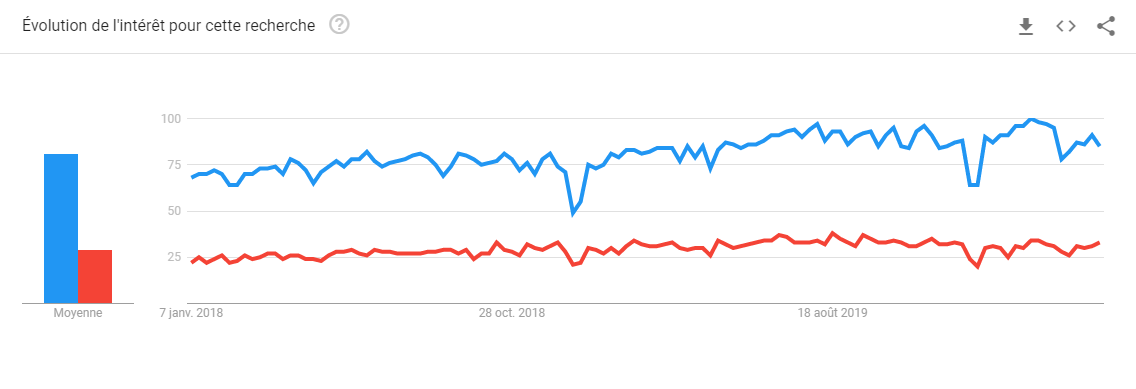
# Analyse d’intégration d’un Framework JS

Pour pouvoir tenter de répondre aux questions établis au point 3.2.1, il faut dans un premier temps décider sur quel Framework cette analyse sera faite.

Pour résumé, deux Frameworks semble viables à ce point : **React.js** et **Vue.js**.

**Vue.js** malgré sont lot d’avantage semble être un Framework encore trop jeune et pas tout à fait stable (encore en beta). Vue.js a été conçu pour des développements de petite taille (one page, applications simple, intégration à la volée…). De plus son faible développement communautaire et sa faible part de marché actuelle ne l’avantage pas.

Pour l’instant (et pour l’année 2020), **React.js** règne sur l’industrie des nombreux Frameworks JavaScripts existants. Sa grande communauté et sa conception axé autour d’applications conséquentes lui donne une place de leader sur le marché.



**29**

**81**

Figure 2 : http://trends.google.com du 01.01.2018 au 30.04.2020

Grâce au points évoqués ci-dessus et à la comparaison faite au point 2.2.4, le choix a été facilité : **React.js** est le Framework choisit pour cette analyse.

## Intégration de React.js à OIH3

React.js fonctionne normalement de pair avec Node.js tant pendant la phase de développement que pendant celle de production. Il est cependant possible, via un chargement de scripts standard, d’intégrer React.js sans Node.js. Ces deux types d’intégrations vont être testés puis approuvé ou rejeté.

Il faut noter que l’intégration de React.js à OIH3 n’est pas « standard » car le backend d’OIH3 est écrit en PHP et non pas en Node.js. A première vue cela ne devrait pas poser de problème majeur, cependant il se peut que notre nouveau frontend n’exploite pas l’entièreté des fonctionnalités, de la force et de la rapidité de React.js.

### Node.js c’est quoi ?

Node.js est une plateforme logicielle libre en JavaScript orientée vers les applications réseau. Elle utilise la machine virtuelle V8, la librairie libuv pour sa boucle d'évènements et implémente sous licence MIT les spécifications CommonJS.

Concrètement, Node.js est un environnement bas niveau permettant l’exécution de JavaScript côté serveur. L’article suivant résume très bien l’histoire du Javascript et l’aboutissement de Node.js : <https://openclassrooms.com/fr/courses/1056721-des-applications-ultra-rapides-avec-node-js/1056866-node-js-mais-a-quoi-ca-sert>

Sur OIH3 côté serveur c’est le langage PHP qui est utilisé et cela va demeurer.

### Pourquoi React.js utilise Node.js ?

Dans la plupart des applications React.js, Node.js assure donc la partie backend du développement. Dans OIH3 nous n’avons pas besoins de cette partie backend puisqu’elle est déjà géré par PHP. Nous cherchons donc dans React.js un remplaçant frontend au même niveau que « smarty » mais intégrant plus de fonctionnalités.

Pour ce faire, notre application React.js devra être compilé avant d’être envoyer en production et c’est là qu’intervient Node.js. En effet Node.js met à disposition une multitude de librairies JS via son gestionnaire de paquets : « [npm](https://www.npmjs.com/) ». Pour faciliter la phase de développement, il est donc conseillé d’utiliser la force de ce gestionnaire de paquets pour développer une application React.js.

### Intégration sans Node.js

Dans un souci de facilité de mise en place et de gain de temps, il parait intéressant d’utiliser React.js sans Node.js. Pour se faire, différents tutoriels sont proposés directement le site de React.js.

* <https://reactjs.org/tutorial/tutorial.html>
* <https://reactjs.org/docs/add-react-to-a-website.html>
* <https://github.com/podio/requirejs-react-jsx>

Grâce à ces tutoriels et quelques aménagements, il est tout à fait possible de faire fonctionner la démonstration d’un composant de base React.js dans OIH3.

Pour se faire, il a fallu importer certains scripts dans le fichier « *header\_new.tpl* » et utiliser la librairie externe suivante : <https://requirejs.org/> permettant de gérer les imports JavaScript. La solution d’intégration trouvée est disponible dans l’archive « M:\OIH3\Documentation\Modules\OmniGED\Documentation\react-without-nodejs.rar ».

Pour valider entièrement cette intégration, il faut pouvoir installer et personnaliser un composant plus complexe. Le but est d’intégrer un composant de type « file-system » pour interagir avec MS Graph et les services d’Office 365. Le composant proposé est le suivant : <https://github.com/imshubhamsingh/file-system-react>

Cette étape fut longue, compliquée et fastidieuse et ce à cause entre autres des « import » ECMAScript 6, de la syntaxe JSX nécessitant une compilation, de l’ajout manuel de toutes les librairies nécessaires…

*Cette solution d’intégration a donc logiquement été stoppé à cette étape et c’est pourquoi une intégration de React.js sans Node.js ne semble pas viable.*

### Intégration avec Node.js

L’intégration de React.js dans OIH3 à l’aide de Node.js est étonnement plutôt simple. Elle a été réalisée dans un tout nouveau dossier : « OIH3/react ». L’arborescence de se nouveau dossier qui contiendra tous les scripts liés à React.js est la suivante :

* *OIH3/react*
  + *assets*

C’est dans ce répertoire que seront ajoutés tous les scripts compilés ainsi que les ressources s’y référant (images, styles…). C’est le seul répertoire qui a besoin d’être téléversé sur le serveur de production.

* + *components*

Dans ce répertoire, chacun des sous-répertoires représente un composant Reac.js fini et intégrable tel quel dans une vue. On peut par exemple y trouver un composant « tic-tac-toe » intégrable n’importe où.

* + *nodes\_modules*

Répertoire des librairies Node.js décrites ci-dessus. Il est nécessaire lors de la compilation. Il ne faut jamais modifier ce répertoire « à la main », mais utiliser les commandes « npm » pour interagir avec ces librairies. Ce répertoire n’a pas besoin d’être versionné.

* + *package.json*

Il s’agit du fichier de « configuration » de Node.js, il est nécessaire pour l’utilisation de « npm ».

* + *webpack.config.js*

Il s’agit du fichier de configuration de compilation du code React.js permettant de créer un serveur de développement et les fichiers de production.

#### Environnement de développement

Pour utiliser l’environnement de développement avec Node.js, il faut commencer par télécharger un installer Node via le lien suivant : <https://nodejs.org/en/>.

On peut ensuite se rendre dans le dossier « OIH3/react » avec le terminal et exécuter les commandes suivantes :

|  |
| --- |
| > npm install |

Cette première commande permet de récupérer toutes les librairies nécessaires au bon fonctionnement des composants React.js. Elle est à exécuter une seule fois au démarrage du projet.

|  |
| --- |
| > npm run-script watch |

Cette seconde commande est à utiliser à chaque fois qu’on environnement de développement veut être utilisé. Elle permet de créer des fichiers scripts de préproduction utilisables par le navigateur.

#### Intégration d’un composant

Pour intégrer un composant React.js dans un fichier smarty «.tpl » rien de plus simple ! Il suffit de copier les 3 lignes suivantes directement à l’endroit souhaité dans le fichier template.

*{\* ---------- React.js start ---------- \*}*<div id="root"></div>  
<script type="text/javascript">*const* ***myBlockData*** = {*$*myBlock.data|@json\_encode};</script>  
<script type="text/javascript" src="/react/assets/bundle/demolist.js"></script>  
*{\* ---------- React.js END ---------- \*}*

1. La première ligne est le point d’attache de notre composant React.js dans le DOM HTML.
2. La constante *myBlockData*, permet de transporter toutes les variables créées dans la fonction PHP « getTemplateData(&$templateDatarray) »
3. Ici le composant invoqué est « *demolist* », cette url est à adapter avec le composant désiré en changeant par exemple *demolist.js* par *tictactoe.js*

#### Mise en production

La dernière étape du développement est la mise en production. Le code doit passer par plusieurs étapes de traitement, de compilation et de minimisation. Fort heureusement, toute ces opérations sont automatisées, il suffit pour ce faire d’exécuter la commande suivante :

|  |
| --- |
| > npm run-script build |

Les scripts React.js compilés seront créés et apparaîtront dans le dossier « OIH3/react/assets/bundles ». Le SCSS sera quant à lui compilé dans le dossier « OIH3/react/assets/css » et les images dans le dossier « OIH3/react/assets/img ».

Cependant, la solution la plus viable reste d’intégrer ces script SCSS et d’utiliser la compilation avec l’architecture des dossiers déjà présentes dans OIH3 (include). Le processus de conserver un seul CSS compilé est important.

#### Conclusion

Cette solution d’intégration des composants React.js avec Node.js en appui lors du développement est fonctionnel et semble viable sur le long terme.

## Adaptation d’un module React.js existant

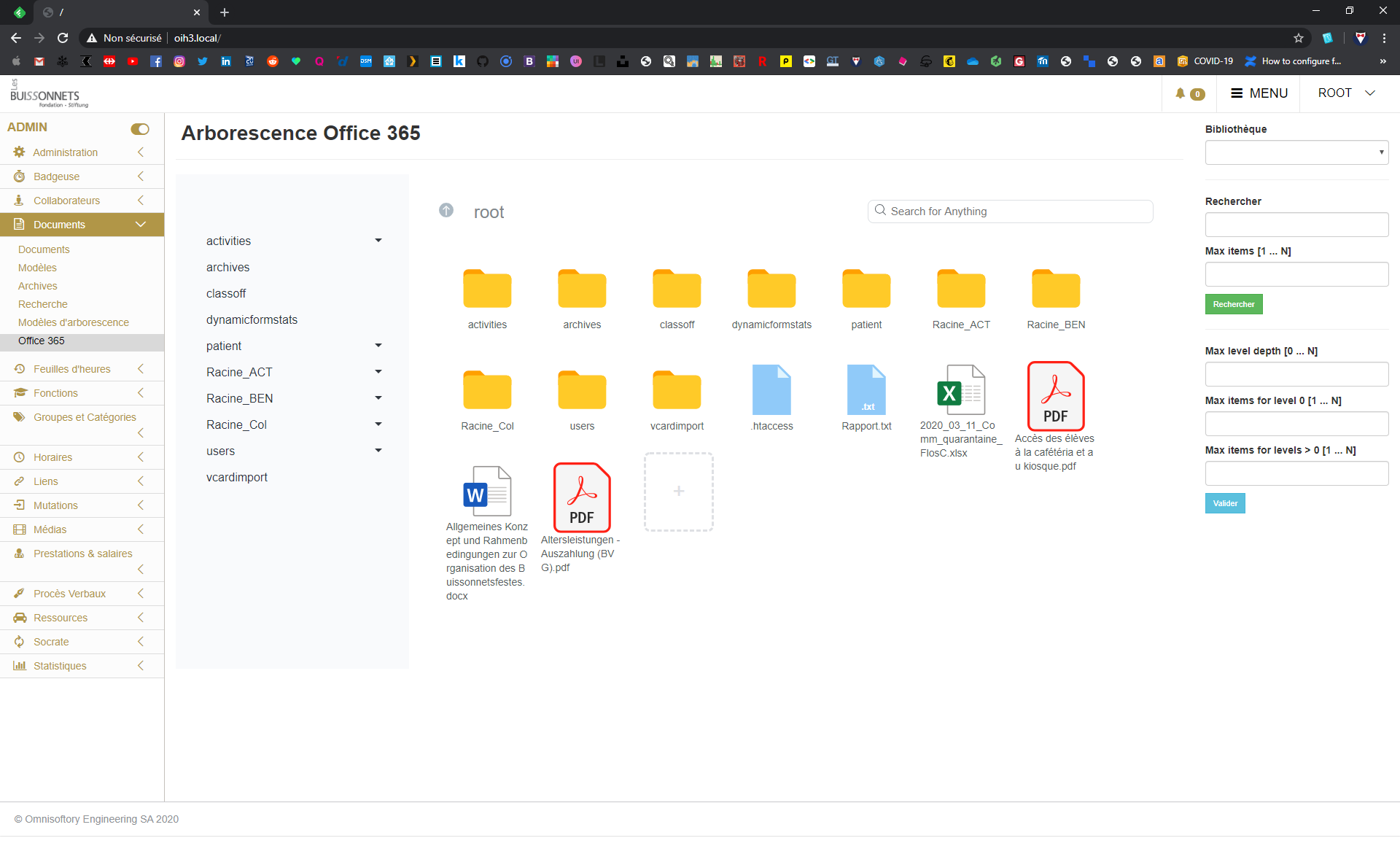
Pour tester et valider l’intégration de React.js dans OIH3, l’idée est d’intégré un composant React.js existant et consistant.

Le module « imshubhamsingh/file-system-react » permettant de créer une arborescence de fichier a été choisi comme base pour ce test. Une démonstration des fonctionnalités présentes dans ce composant est disponible à l’adresse suivante : [file-system-react](https://camo.githubusercontent.com/8caf9181bc5c685398fc75d3d585d5db4fe1cf3b/68747470733a2f2f7265732e636c6f7564696e6172792e636f6d2f67726f6f6f742f696d6167652f75706c6f61642f76313535353231353236312f736d616c6c636173652f323031392d30342d31345f30392e32342e31312e676966).

### Arborescence racine

L’intégration de ce composant React.js a été faite dans le module « Document » sous le menu « Démo React.js ». Elle reprend l’arborescence présent dans le menu « Office 365 » et reflète celle présente sur MS SharePoint.

Celui-ci permet d’avoir une arborescence de fichiers sur la gauche et une navigation type « desktop » au centre. Un système de recherche est déjà intégré ainsi que des fenêtres de propriétés des documents.



Le module React.js remplace ici le template smarty dans la zone encadrée. Le résultat est plutôt concluant et s’adapte bien à OIH3.

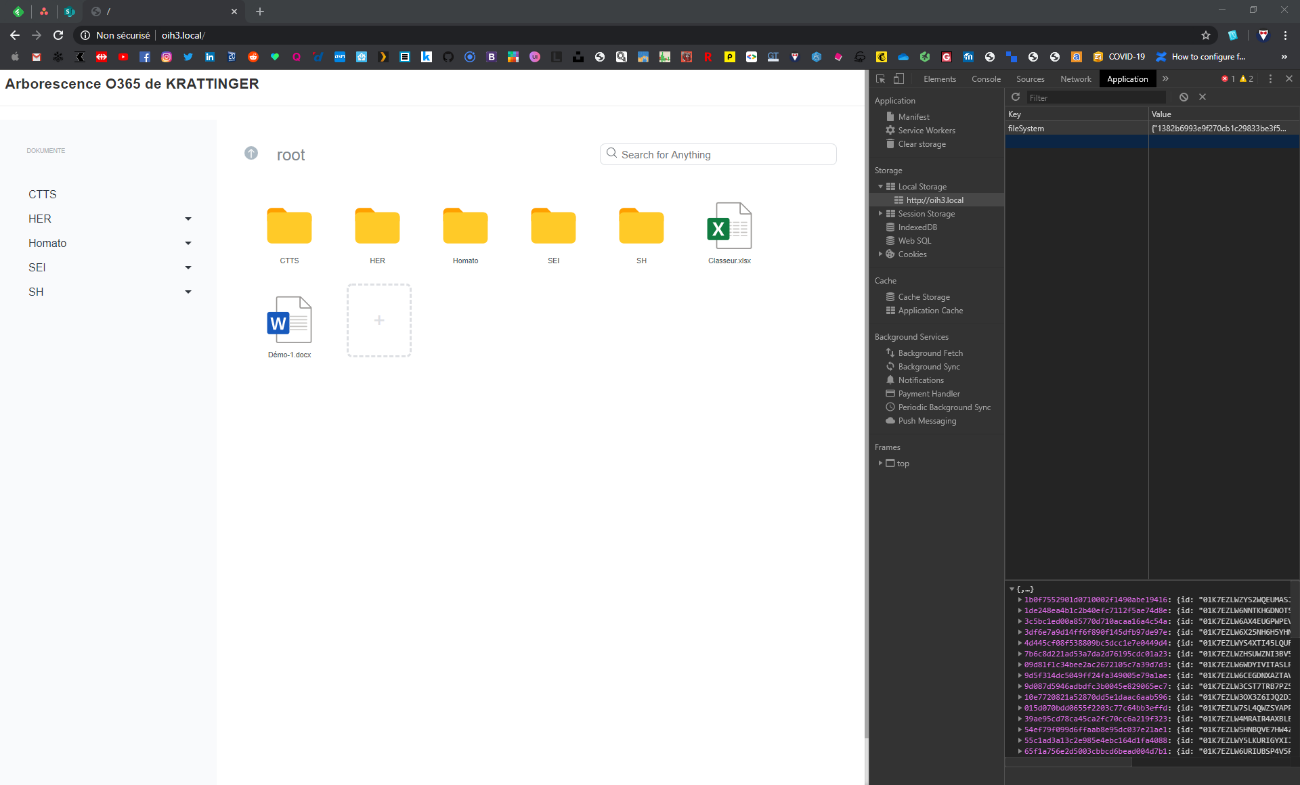
Pour arriver à ce résultat d’affichage encapsulé, il a fallu adapter le module. Cette étape est assez conséquente car il faut maitriser la syntaxe React.js puis comprendre le développement et la structure de module lui-même.

#### Maquette live

La maquette de test en live de cette implémentation est disponible à l’adresse suivante : <https://omniged.ch/index.php?dom=admin&op=loadmod&mod=download&subOp=reactjs&action=none>. Cette maquette est d’une lenteur atroce et plusieurs bugs sont à corriger.

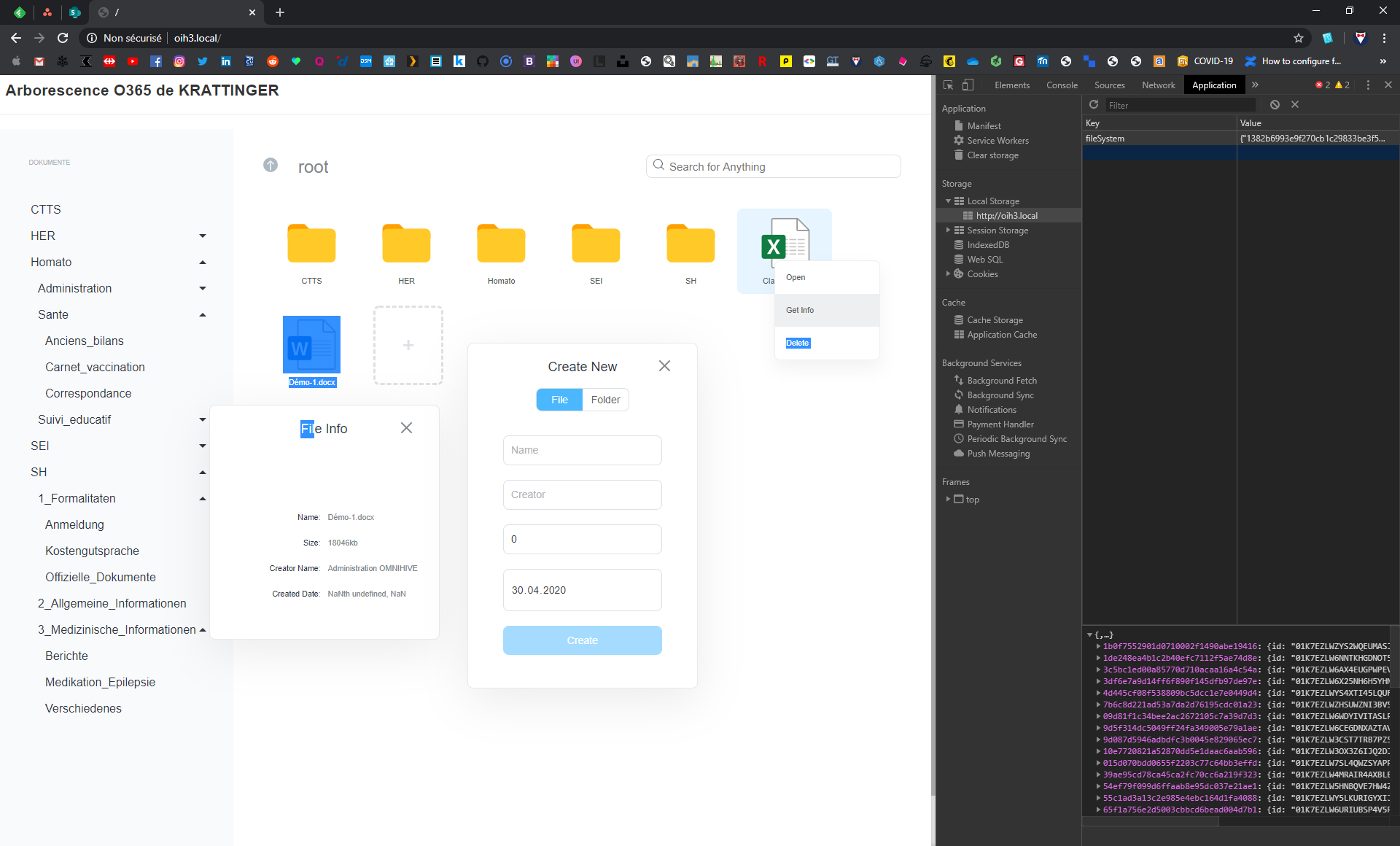
### Arborescence spécifique

Une fois l’arborescence principal mise en place, l’intérêt de de pouvoir réutiliser ce composant à différents endroits de OIH3. Par exemple pour un patient, l’interface intégré fonction plutôt correctement.

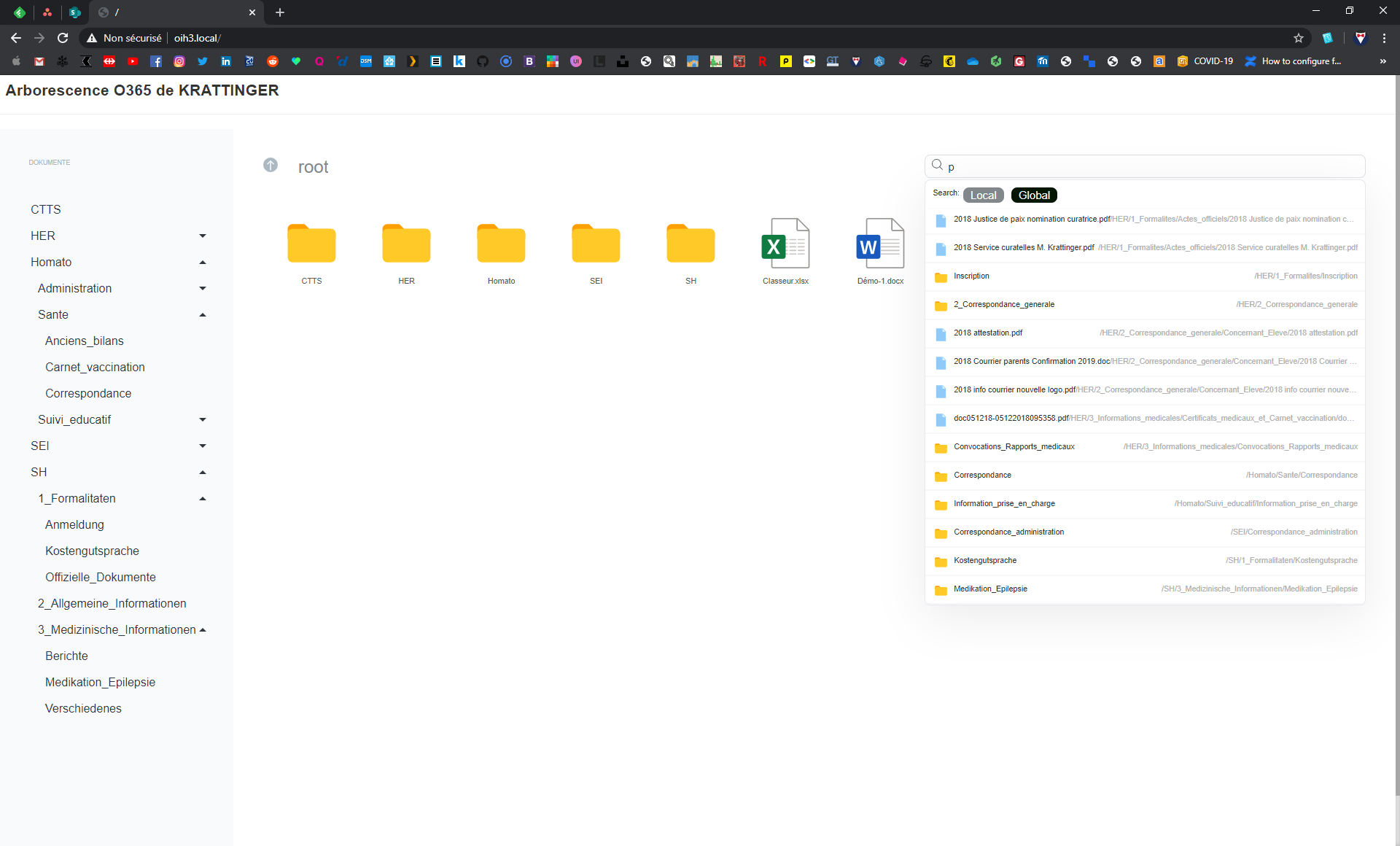


Plusieurs fonctionnalités sont déjà présente comme le clic droit permettant d’afficher un menu d’options. Ce menu permet alors d’ouvrir directement le fichier sur Office 365, d’afficher les informations du fichier ou de le supprimer.

Il y a également à disposition une fenêtre permettant de créer des fichiers à la volées.



Le champ de recherche permet d’effectuer directement une recherche sur l’arborescence local uniquement en utilisant React.js. Cette recherche à l’avantage d’être extrêmement rapide. Il serait envisageable de modifier ce champ de recherche pour y intégrer la recherche plein texte via MS Graph.



#### Maquette live

Cette maquette bien que fonctionnant avec tous les patients est extrêmement lente. Pour tester les fonctionnalités de ce composant React.js, il est conseillé d’utiliser le patient préchargé suivant : « <https://omniged.ch/index.php?dom=admin&op=newentry&mod=download&subOp=reactjs&action=none&vid=5714> ».

## Développement à envisager

Le composant React.js proposé par « imshubhamsingh/file-system-react » correspond à une bonne base pour créer une nouvelle interface GED. Cependant pour qu’il soit entièrement fonctionnel avec OIH3, il va falloir développer les fonctions requises définit au point 1.2.

Cela semble envisageable, le temps nécessaire pour y parvenir est en revanche conséquent. En effet, bien que le module de base soit présent et utilisable, il faut apprendre la syntaxe de React.js pour pouvoir y apporter les modifications désirées.

Dans le temps imparti, il faut définir des fonctionnalités prioritaires et les développer. L’intégration de React.js nous permet tout à fait de produire des composants de façon incrémentale.

## FAQ

Différents points méritent encore une attention toute particulière pour confirmer ou infirmer la direction choisit.

### Réutilisation des composants

La réutilisation des composants est un point essentiel dans OIH3. Avec l’intégration de React.js proposé ici, cette réutilisation semble tout à fait possible si le développement de chacun des composant est réalisé dans cette optique.

Le fait de pouvoir passer des données brutes entre PHP et React.js et de les interpréter comme désirée tant du côté backend que frontend est un avantage certain.

### Traductions

Les traductions nécessaires au bon fonctionnement de l’interface sont toujours gérées par le PHP. Elles peuvent ensuite être « transporté » comme n’importe qu’elle autre variable vers React.js au moyen du bloc de données « myBlockData ».

En outre si certaines traductions nécessitent un affichage tardif, il est tout à fait envisageable d’effectuer des appels AJAX pour les récupérer et les traiter directement dans un composant React.js.

### Avantages et plus-value

Les avantages d’intégrer un Framework JS comme React.js sont multiples. Au travers de ce fichiers plusieurs ont déjà été abordées.

On peut notamment citer un gain de temps, une interface plus agréable à utiliser pour les utilisateurs, la possibilité d’intégrer de multiples librairies existantes, un développement de la partie frontend structuré…

### Compatibilité avec différents OS / Navigateurs

L’intégration réalisé à actuellement été testé et est fonctionnelle sur les dispositifs suivants :

* + Windows 10.1903 / Google Chrome version 81.0
  + Windows 10.1903 / Firefox
  + Windows 10.1903 / Microsoft Edge version 81.0.416.68
  + macOS 10.15 / Google Chrome version 81.0.4
  + macOS 10.15 / Safari version 13.1
  + iOS 13.4.1 / Safari version 13.1 (clic droit à adapter)

A tester :

* Android

### Performance

A ce stade, il est difficile de répondre à cette question de gain ou de perte de performance. Cependant si le Framework JS choisit est intégré correctement en respectant les standards et les contraintes techniques, il ne devrait pas y avoir d’énorme problèmes de performance.

Liens utiles

[https://fr.wikipedia.org/wiki/Ext\_(JavaScript)](https://fr.wikipedia.org/wiki/Ext_(JavaScript)

<https://www.slant.co/options/35/alternatives/~extjs-alternatives>

<https://2019.stateofjs.com/front-end-frameworks/>

<https://vuejs.org/v2/guide/comparison.html>

<https://programmingwithmosh.com/javascript/react-vs-vue-a-wholesome-comparison/>

<https://www.monterail.com/blog/vue-vs-react-2019>

<https://medium.com/js-dojo/react-vs-vue-exact-todo-app-comparison-by-example-14cc56efc5e5>

<https://medium.com/@davisonpro/an-advanced-guide-on-setting-up-a-react-and-php-web-app-acaedb21ab3a>

Abréviations

Abréviations utiles à tous les lecteurs :

|  |  |
| --- | --- |
| Abréviation | Description |
| GED  GUI | Gestion électronique des documents  Interface graphique (Graphical user interface) |

Historique

Ci-dessous, vous pouvez suivre l’historique des modifications apporté au présent document.

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Auteur | Version | Commentaire |
| 13/01/2020 | T.A. | 1.0 | Première version révisée |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |