# Rapport de projet

# JXS / JXW

*"J'atteste que ce travail est original, qu'il indique de façon   
appropriée tous les emprunts, et qu'il fait référence de façon   
appropriée à chaque source utilisée''*

Florent Catiau-Tristant

Alexis Brault

Yoann Boyère

ESIR 2 Systèmes d’Information

2016 - 2017

***Sommaire***

[I) Outils utilisés 3](#_Toc481772577)

[1. Outils de gestion / communication 3](#_Toc481772578)

[2. Outils techniques 3](#_Toc481772579)

[II) Structure du projet 4](#_Toc481772580)

[1. Architecture générale 4](#_Toc481772581)

[2. Organisation du code 4](#_Toc481772582)

[III) Fonctionnement du serveur REST 5](#_Toc481772583)

[IV) Manuel d’utilisation 6](#_Toc481772584)

[1. Connexion 6](#_Toc481772585)

[2. Parcours des fichiers 7](#_Toc481772586)

[3. Fonctions générales 8](#_Toc481772587)

[4. Modification de fichier 8](#_Toc481772588)

[V) Fonctionnalités présentes 10](#_Toc481772589)

[VI) Difficultés rencontrées 10](#_Toc481772590)

### Outils utilisés

#### Outils de gestion / communication

Comme pour tout projet, nous avons été obligé de nous organiser. Nous avons donc utilisé Trello afin de pouvoir séparer les différentes tâches à effectuer et lister les bugs à corriger. Nous avons utilisé aussi BitBucket (et donc Git) en tant que gestionnaire de version de notre code source.

#### Outils techniques

Pour le coté client, notre choix s'est très vite porté sur **Angular4**. Cette technologie nous semble très intéressante à utiliser car elle est largement utilisée dans le monde. De plus, elle permet une factorisation de code non négligeable. Nous utilisons **Bootstrap** **3** pour le design de l’application couplé à la bibliothèque d’images [material.io](https://material.io/icons/#ic_dashboard) fournie par Google.

Du côté du serveur, nous avons choisi d’utiliser node.js pour la simple et bonne raison que c'est très simple d'utilisation et performant. Qui plus est, nous avons eu une présentation en cours de JXS qui nous a donné envie d'approfondir le sujet.

### Structure du projet

#### Architecture générale

Le client dispose d’une interface graphique claire suivant les principes de l’UX Design. Toutes les actions sur les fichiers sont appelées sur le serveur via des requêtes http « GET » ou « POST ».

Le serveur sert d’interface entre le client et les API REST des différents cloud. Chaque action est une requête sur les différentes api. De cette manière, le client n’a pas à se soucier des paramètres des requêtes.

Lors de la récupération des fichiers, le serveur obtient l’arborescence des fichiers présents dans chaque source au format Json. Puis il se charge de fusionner les deux dans un format json unifié, qu’il envoie ensuite au client. Tous les fichiers de l’arborescences sont alors présents.

#### Organisation du code

Le client se découpe en plusieurs composants majeurs. Le premier constitue l’affichage statique de l’application (header et footer). Puis d’autres composants ont pour rôle d’afficher les différentes pages de l’application. En l’occurrence, il s’agit des pages de connexion et des fichiers. Nous utilisons le système de routage d’Angular pour afficher ces composants (/home pour le premier et /files pour le second). D’autres composants plus petits représentent des modules comme la connexion à un cloud ou l’affichage du taux de remplissage d’un cloud par exemple.

Nous disposons de deux classes qui servent de modèle : *FileDrive.ts*, qui représente un fichier de l’application et *api.service.ts* qui est le service utilisé par notre application pour faire le lien avec le serveur.

Coté serveur, le fichier *server.js* implémente tous les points d’accès du serveur[[1]](#footnote-2). Chaque fonctionnalité est redirigée vers un objet « connector.js » qui lui implémente les requêtes exactes vers les api des cloud disponible. Pour le moment, nous avons donc deux connectors, « *GoogleDriveConnector.js* » et « *DropboxConnector.js*». Ils ont tous deux le même format, et auraient pu implémenter une même interface si javascript l’avait permis.

### Fonctionnement du serveur REST

Comme expliqué précédemment, le serveur REST constitue le point de liaison entre le client et les api des services cloud. Le fichier *server.js* indique tous les points d’accès de l’application. A la fin de l’url http://localhost:8080/ il suffit d’ajouter les paramètres suivants :

* connect/<cloud>
* listFiles/
* spaceUsage/
* accountInfos/
* addNewFolder/<cloud>?name=<name>&path=<path>
* move/<cloud>?from\_path=<from>&to\_path=<to>
* rename/<cloud>?id=<id>&name=<name>
* delete/<cloud>?id=<id>

Certaines commandes ne nécessitent pas de préciser sur quel cloud l’effectuer, le serveur dispose d’une fonction générale qui lance la même fonction sur chaque cloud à la fois. Par exemple pour récupérer les fichiers de Google Drive, il faut lancer /listFiles/GoogleDrive. Mais si l’on souhaite les fichiers de toutes les sources confondues, il faut lancer la commande /listFile/. Dans ce cas, le serveur fera la fusion des fichiers.

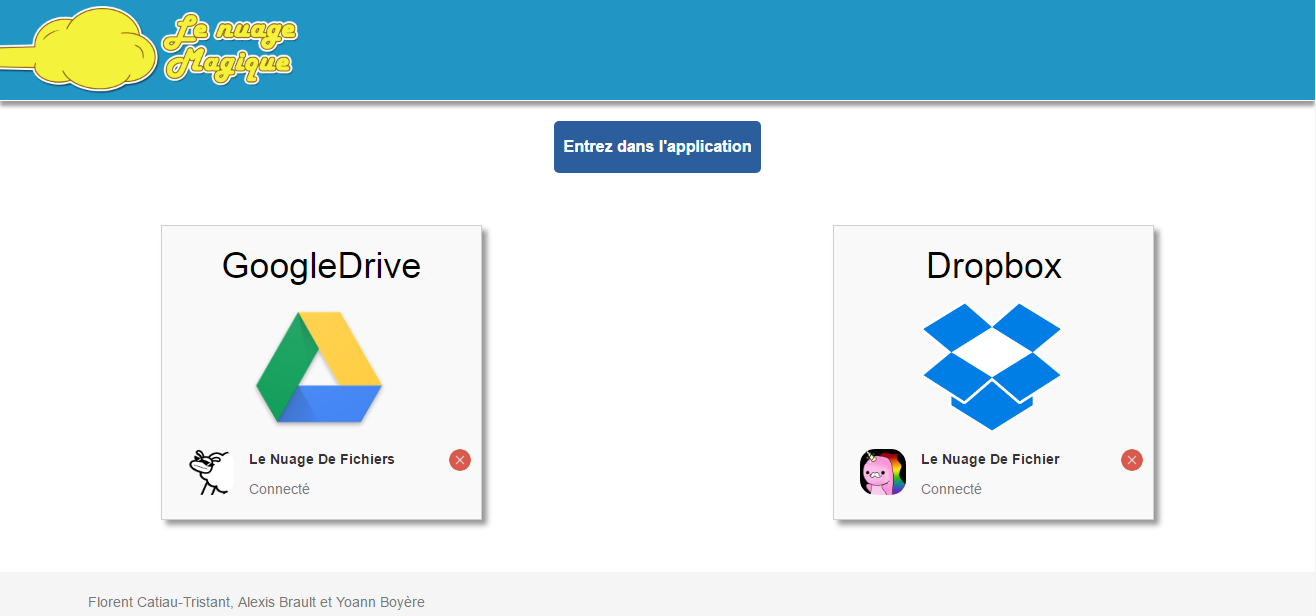
Comme indiqué, certaines commandent nécessitent des paramètres dans la requête. Nous avons indiqué ici les paramètres utilisés dans la dernière version du serveur qui n’a pas été totalement implémentée. Cette version permet d’utiliser de la même manière les fonctions pour chaque cloud.

En l’occurrence, pour créer un répertoire, il faut préciser le « path » de celui-ci sur Dropbox, tandis qu’il faut l’id du répertoire parent pour GoogleDrive. En version 1, les paramètres des requêtes étaient donc différents.

### Manuel d’utilisation

#### C:\Users\Florent\Pictures\ShareX\Screenshots\2017-05\vivaldi_2017-05-05_16-16-59.pngConnexion

L’application s’ouvre sur cette page de connexion, où sont affichés les différents services cloud supportés.

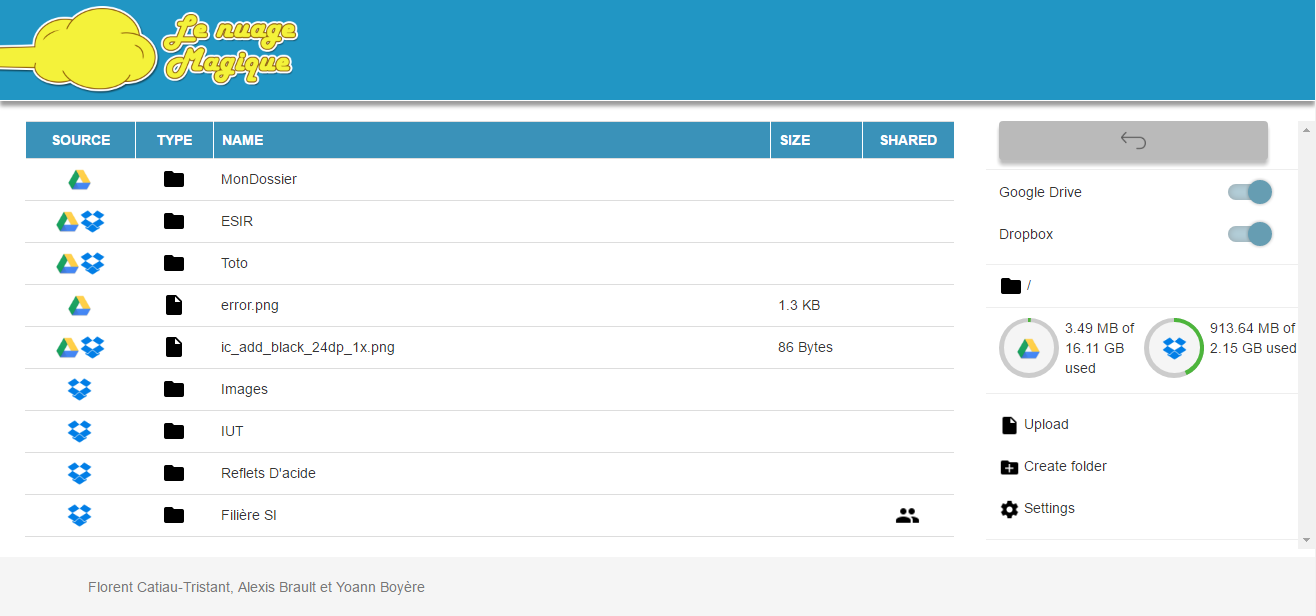
Les boutons de connexion redirigent vers les pages sécurisées des services eux-même. Dans certains cas, il se peut que la connexion se fasse directement si le navigateur a enregistré la connexion en cache.

Une fois connecté, quelques informations du profil sont affichées afin de vérifier directement quels sont les comptes utilisés. Il est par ailleurs possible de se déconnecter via la croix rouge.

#### Parcours des fichiers

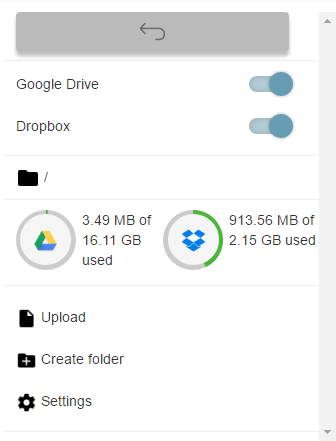
Il suffit de cliquer sur le bouton « Entrer dans l’application » pour accéder aux fichiers. Après un temps de chargement, les fichiers apparaissent dans un tableau.





Pour accéder à un dossier, il suffit de double cliquer sur celui-ci. Un fichier peut être sélectionné par un clique simple.

#### Fonctions générales

1. Bouton de retour. Il sert à revenir au dossier précédent. Il est désactivé lorsqu’il est impossible de revenir en arrière.

**2.**

**1.**

1. Ce sont des filtres d’affichage. Ils permettent de n’afficher que les fichiers de certains services cloud connectés.

**3.**

1. Affichage du chemin courant depuis la racine.

**4.**

1. Taux de remplissage des différents services cloud connectés.

**5.**

1. Fonctions d’ajout de fichier, de création de répertoire vide et de paramètres. Les paramètres redirigent vers la page d’accueil afin de connecter ou déconnecter d’autres services cloud.

#### C:\Users\Florent\Pictures\ShareX\Screenshots\2017-05\vivaldi_2017-05-05_18-17-53.pngModification de fichier

Ces fonctionnalités apparaissent lorsqu’un fichier est sélectionné.

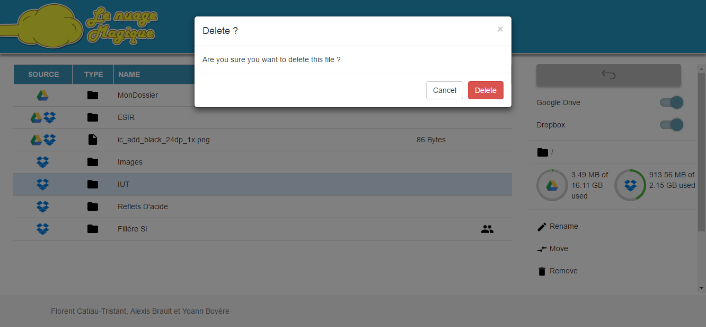
1. Renommer un fichier
2. Déplacer un fichier. Il faut indiquer le chemin du nouveau dossier sans oublier de noter à la fin du chemin le nom du fichier.

**1.**

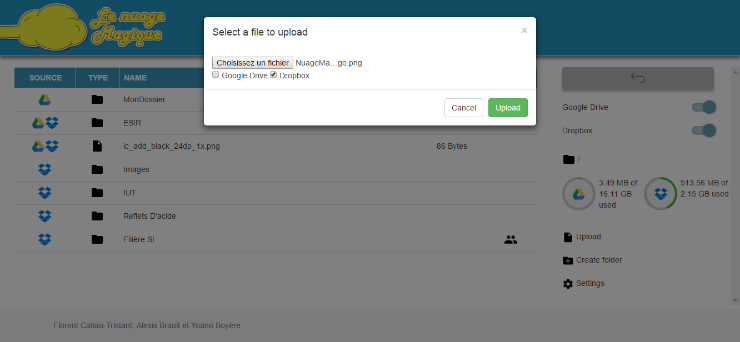
**2.**

1. Supprimer un fichier.

**3.**



A chaque action, une pop-up apparait à l’écran. Il faut alors renseigner les valeurs nécessaires et confirmer l’action.



Il est en général possible de choisir pour quel service cloud l’action doit s’effectuer dans le cas de fichiers présent sur plusieurs sources.

### Fonctionnalités présentes

|  |  |  |
| --- | --- | --- |
| **Fonctionnalités** | **Google Drive** | **Dropbox** |
| Explorateur de fichier unique |  |  |
| Fusion de dossiers de même nom |  |  |
| Upload de fichier |  |  |
| Déplacer un fichier |  |  |
| Renommer un fichier |  |  |
| Supprimer un fichier |  |  |
| Affichage des détails d’un fichier |  |  |
| Partage de fichier |  |  |
| Mise en cache des données |  |  |
| Filtre des dossiers |  |  |
| Spinner affichant le taux de remplissage |  |  |

### Difficultés rencontrées

Pour le serveur, la principale difficulté a été de comprendre et d’utiliser correctement les api REST des services cloud sans passer par leurs api Javascript déjà fournies.

Du coté client, malgré l’utilisation d’Angular qui facilite grandement les choses, il est assez difficile de rendre l’application dynamique à l’ajout de services cloud. En l’état, il faut rajouter beaucoup de composants et les connecter manuellement lorsqu’un service cloud est ajouté. Par exemple, même si l’on ne se connecte pas à Dropbox, les composés liés comme l’affichage de l’espace disponible sont affichés.

1. Voir partie III) Fonctionnement du serveur REST [↑](#footnote-ref-2)