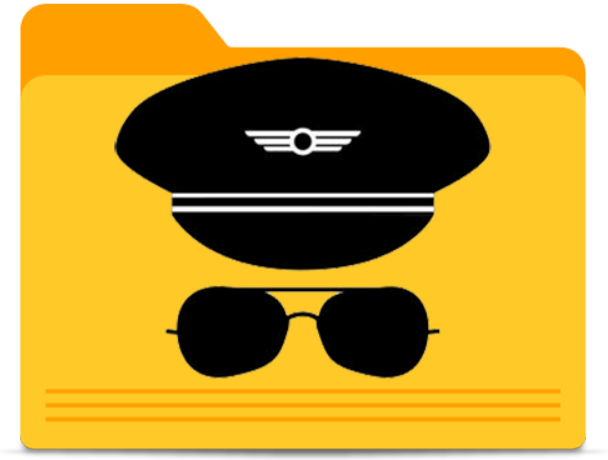
Projet PNSInnov



Équipe M - **Pilot**

Nassim BOUNOUAS, Joël CANCELA VAZ, Loïc GARDAIRE, Johann MORTARA

SI4

15/06/2018

Sommaire

I. Description du projet

a) Présentation de l'innovation

Étant utilisateurs de plateformes de stockage en ligne comme *Google Drive* ou *Dropbox*, nous avons pu établir le constat suivant : il est aujourd'hui difficile de conserver un espace de stockage en ligne ordonné. En effet, si l'on souhaite classer nos fichiers, notre seule solution à ce jour est d'effectuer cette opération à la main, opération pouvant rapidement devenir très chronophage et répétitive.

Pilot est une solution qui aide à la gestion d'un espace de stockage en ligne. Notre solution se greffe sur différents espaces de stockage en ligne déjà existants. Elle est innovante dans la mesure où elle offre à l'utilisateur la possibilité de ranger automatiquement ses fichiers en fonction de règles établies et de les chiffrer s'il le désire. Un ensemble de règles prédéfinies est déjà présent et couvre les besoins les plus courants. Cependant, si l'utilisateur a des besoins plus précis, il est en mesure d'en établir de nouvelles.

Pour clarifier, une règle se définit par un ensemble de conditions qui, si elles sont respectées par un fichier, déclenchent un ensemble d'action sur ce dernier, l'action la plus simple étant un déplacement mais pouvant tout aussi bien être une suppression, un renommage ou encore un chiffrement. Les conditions dans notre cas sont un motif dans le nom, une extension de fichier ou le type MIME. D'autres conditions sont envisagées, notamment sur les métadonnées des fichiers, par exemple le lieu ou la date de prise d'une photo.

b) État de l'art

Le tableau ci-dessous présente une comparaison entre notre solution et les potentiels concurrents directs ou indirects présents sur le marché actuellement. L'objectif est d'analyser point par point les fonctionnalités présentes dans notre système et de les comparer à celles des concurrents afin de nous distinguer.

	Pilot	*Hazel*[1]	*Funnel*[2]	*Zapier*[3]
Rangement de fichiers par règles	Oui	Oui	Oui (des catégories de fichiers qui regroupent des extensions)	Oui
Règles personnalisables	Oui	Oui	Oui	Oui
Support *Google Drive*	Oui	Non	Oui	Oui
Support *Dropbox*	Oui	Non	Oui	Oui
Chiffrement	Oui	Non	Non	Non
Limitations		Limité à Mac, en local uniquement	Pas de possibilité d'ajouter de types de fichiers aux catégories. *Application abandonnée et indisponible.*	Permet le déplacement périodique de fichiers un par un, création des règles beaucoup plus complexe

Figure n°1 : Tableau comparatif des possibles concurrents

c) Vision à 3 semaines

Dans le cadre de ce projet, nous avions 3 semaines pour réaliser un produit viable qui puisse démontrer une véritable innovation et sache capter l'intérêt des utilisateurs. Nous avons donc sélectionné les objectifs suivants, qui apportent le plus de valeur :

- Le rangement des fichiers assuré par un moteur de règles opérationnel avec au moins trois types de conditions (motif dans le nom de fichier, extension et type MIME).
- La visualisation de l'arborescence des fichiers des systèmes de stockage afin de s'abstraire des visualisations de chaque plateforme et pouvoir facilement passer de l'une à l'autre.
- La création, l'affichage et l'ordonnancement de règles personnalisées.
- Une simulation de l'application des règles sur les fichiers afin de prévisualiser le résultat obtenu sur le système de stockage.
- Le chiffrement et le déchiffrement d'un fichier, ainsi que son téléchargement et son téléversement.
- La connexion avec au moins deux plateformes de stockage différentes afin de démontrer l'interopérabilité de notre solution.

[1] https://www.noodlesoft.com
[2] http://www.awegoinc.com/filefunnel/filefunnelscreenshots.html
[3] https://zapier.com/blog/organize-files-folders

Le découpage de l'implémentation de ces fonctionnalités pour les trois semaines est le suivant :

La première semaine, l'objectif principal était d'avoir un produit minimum viable. Nous avons donc commencé par intégrer l'API de *Google Drive*, nous permettant d'accéder aux fichiers stockés sur ce dernier. La seconde étape fut de mettre en place le moteur de règles. Il était alors minimal et permettait seulement d'appliquer des règles stockées statiquement dans un fichier. Les règles permettent un classement via l'extension et le type MIME du fichier. Pour avoir un visuel de l'arborescence du *Drive*, nous avons implémenté un affichage de ce dernier, permettant de voir les modifications après application des règles.

La deuxième semaine, nous avons ajouté le troisième type de condition avec la détection de motifs dans le nom des fichiers. Le motif peut être trouvé en début ou en fin de nom, ainsi qu'être contenu dans le nom du fichier. La saisie du motif, bien que reposant sur le principe d'expression régulière, est effectuée par le biais d'un formulaire pensé pour l'utilisateur néophyte. Dans la même optique, nous avons donc fait évoluer les règles en ajoutant la possibilité pour l'utilisateur de créer ses propres règles, ainsi que de les visualiser sur une page dédiée. Pour finir, la gestion de multiples utilisateurs a été ajoutée à notre système afin de faire face aux difficultés de cloisonnement que ce dernier apporte.

La troisième semaine, nous souhaitions faciliter l'utilisation de notre système par l'utilisateur en ajoutant un simulateur d'application des règles. Ce dernier exécute les règles de l'utilisateur dans un contexte de *bac à sable* et affiche l'arborescence résultante. Il n'y a pas de modification du *Drive* ; cela permet à l'utilisateur de prévisualiser le résultat de l'application des règles. Du côté des règles, l'utilisateur peut, à présent, classer ses règles personnelles pour décider de l'ordre dans lesquelles elles s'appliqueront. Enfin, nous avons ajouté un nouvel espace de stockage : *Dropbox*, nous permettant de commencer l'abstraction nécessaire pour pouvoir devenir interopérable avec les *Drives* disponibles sur le marché.

d) Vision à long terme

Nous souhaitons, à long terme, proposer une solution de stockage embarquée avec notre application afin de se présenter comme une alternative aux solutions en ligne initialement utilisées. En parallèle, *Pilot* resterait opérationnel sur *Google Drive*, *Dropbox* et *OneDrive*. Parmi les autres possibilités évoquées, le projet *Syncthing* cité dans la partie technique pourrait être intégré en tant que couche de stockage.

Une des fonctionnalités majeures est de proposer une création de règles accessible à n'importe quel utilisateur sans qu'aucune connaissance technique ne soit requise. Une valeur ajoutée à cette création de règles est la suggestion de règles déduites des informations extraites de l'arborescence des fichiers de l'utilisateur. Par exemple, si l'utilisateur possède un dossier « *Mes photos* », le système lui suggère de créer une règle qui déplace les fichiers ayant une extension JPEG ou encore une extension associée au format RAW dans ce dossier.

En parallèle d'une création simplifiée de règles par l'utilisateur, nous souhaitons apporter la possibilité de créer des sous-arborescences lors de l'application des règles. Actuellement, les règles permettent uniquement de créer des dossiers ou de déplacer les

fichiers dans des dossiers situés à la racine du *Drive*. Nous voulons pouvoir créer des arborescences dynamiquement, basées par exemple sur un format de date. Prenons le cas d'un tri de photos : nous aurions alors un premier dossier représentant l'année, contenant des sous-dossiers pour chaque mois permettant ainsi de classer ces fichiers dans une arborescence selon le mois et l'année.

La protection de la vie privée est déjà présente au moyen du chiffrement AES dans le produit livré à l'issue de ces 3 semaines. À long terme, nous souhaitons offrir des fonctionnalités annexes telles que le choix de l'algorithme de chiffrement si cela se révèle pertinent pour l'utilisateur, l'importation et l'exportation de la clé de chiffrement ou encore le partage de fichiers chiffrés au moyen d'une clé partagée entre les personnes ayant accès à un fichier chiffré et partagé.

II. Choix techniques

a) Première vision du produit

Initialement et suite aux échanges avec notre sponsor, nous avions envisagé de construire notre système par-dessus un projet open-source de synchronisation de fichiers. Cette solution, *Syncthing*[4], présente le principal élément que nous défendions sur notre sujet initial (à savoir la synchronisation décentralisée de fichiers) et nous souhaitions donc nous en servir comme support de stockage sur lequel opérer notre système de rangement.

Nous avons rapidement écarté cette possibilité, une première prise en main de la solution nous ayant révélé une API très limitée. Nous avons envisagé de travailler directement sur le projet *Syncthing* mais celui-ci est intégralement codé en Go (85.6% du code source en juin 2018, les 14.4% restants ne correspondant qu'à l'interface homme-machine du projet en HTML / Javascript). Aucun d'entre nous n'a pour le moment d'expérience suffisante en Go et un essai rapide de modification du projet nous a convaincu de ne pas prendre le risque de nous engager dans cette voie.

b) Choix de la pile technologique

Le choix de la pile technologique a été important dans la démarche de notre projet. Nous voulions obtenir un produit minimum viable le plus rapidement possible afin d'apporter le plus de valeur pour le client. Nous avons donc opté pour une technologie avec laquelle nous étions le plus à l'aise afin que celle-ci ne soit pas un frein à notre progression. Néanmoins, nous ne voulions pas négliger le fait que notre projet devait être maintenable même après ces trois semaines.

Nous avons choisi d'utiliser le framework *Jersey* afin de rester dans l'écosystème Java et permettre une coopération accrue avec les autres technologies Java EE telles que *EJB*, *JPA* ou encore *JAXB*. Ce framework permet de produire rapidement et facilement des services *RESTful*, ce qui nous a permis de viser en priorité la valeur client. De plus, les implémentations clientes

[4] https://syncthing.net

des différents *Drives* sont toutes disponibles en Java, et un de nos membres connaissait déjà bien le framework.

Nous pouvons renforcer cet argument par la présence native dans Java d'un système de chiffrement nous permettant d'implémenter notre axe de sécurité.

La notion de route très présente au sein de *Jersey* s'est avérée utile notamment dans la gestion du protocole *OAuth* nécessitant des *callbacks* sur notre application afin de pouvoir accéder en délégation au *Drive* de l'utilisateur.

Un autre choix fort de notre implémentation a été de ne pas séparer (en terme d'architecture) le front-end et le back-end. Nous avons remarqué lors des précédents projets que nous sommes parfois tentés de passer sur de l'interface lorsque nous sommes confrontés à des difficultés du côté back-end, ce qui a pour conséquence de nous rendre moins productif sur les fonctionnalités qui ont le plus de valeur pour le client. Nous nous sommes donc tournés vers le framework *JSF* permettant de facilement nous connecter à notre *back-end* Java tout en ayant une implémentation MVC liée à *Jersey*. *JSF* était donc tout indiqué pour notre projet. De plus, tous les membres du projet avaient déjà utilisé cette technologie auparavant nous permettant ainsi de ne pas perdre de temps à apprendre une nouvelle technologie.

c) Choix du moteur de règles

Nous avons fait le choix d'utiliser le moteur de règles *Drools*, aussi nommé *JBoss Rules*, lui aussi en Java. Ce moteur de règles nous a initialement été suggéré par notre sponsor. Ce choix se justifie par plusieurs raisons :

- La volonté de séparer la création et la gestion des règles du reste du code, ce qu'un simple *switch case* ne permettrait pas.
- L'expressivité des règles de *Drools* nous facilite la future création d'un langage dédié pour proposer par exemple de la composition de règles. Drools utilise son propre format de règles contrairement à la plupart des autres moteurs de règles qui utilise un format dérivé du Lisp, ce qui nous permet de générer ces fichiers plus facilement pour la création dynamique des règles propres à chaque utilisateur.

```
1: package org.drools;
2: rule
3:   when
4:     $file:FileInfo(acceptedExtensions)
// Si l'extension du fichier correspond à une extension supportée
5:   then
6:     $file.moveFile(folderName);
// Le fichier est déplacé vers un dossier folderName
7: end
```

Figure n°2 : Exemple de règle en Drools

Jusqu'à présent *Drools* satisfaisait nos besoins, mais si à l'avenir nous devions passer à l'échelle, il se pourrait que *Drools* ne suffise plus pour assurer des performances acceptables.

Nous pourrions, a posteriori, envisager de passer sur des alternatives à *Drools*, à savoir : *Corticon*, *OpenRules* ou encore *IBM ODM*, qui sont réputés plus performants mais sont des solutions payantes. Notre comparaison actuelle des différents moteurs reste assez superficielle et repose en grande partie sur leurs documentations disponibles en ligne. Une prise en main directe de ces solutions s'avérera nécessaire lors d'une potentielle mise en production et passage à l'échelle.

d) Choix du chiffrement

Notre implémentation actuelle du chiffrement relève plus de la preuve de concept que du produit final. Nous avons fait le choix d'un algorithme unique, à savoir AES, dans son implémentation présente au sein de Java. Cette version ne supporte pas de clés supérieures à 16 bits, ce qui est très faible. Les versions supportant des clés de longueur supérieure (longueur 256 bits recommandée par la NSA) sont aisément utilisables mais nécessitent une bibliothèque externe. Nous avons choisi AES car il est approuvé par la NSA, c'est un des algorithmes les plus sécurisés et c'est un algorithme à clé symétrique ce qui nous suffisait amplement pour notre produit minimum viable.

e) Difficultés rencontrées

Google Drive, comme *Dropbox*, a une implémentation cliente assez difficile à prendre en main, la documentation existante manquant le plus souvent de clarté. Ce problème est amplifié par la disponibilité de différentes versions des API et de la représentation des requêtes sous forme d'objets conçue de manière peu intuitive.

Prenons un cas concret : lors de l'implémentation de notre visualisation de *Google Drive* dans notre système, il nous a fallu recréer l'arborescence de ce dernier. Nous avons donc commencé par chercher une méthode nous permettant de récupérer les *enfants* d'un dossier. Cette méthode existe mais elle est dépréciée dans la version actuelle de l'API. Aucune méthode ne la remplace, excepté une méthode permettant de récupérer l'ID du dossier *parent*. L'algorithme de création de l'arborescence aurait donc pu être grandement simplifié si la méthode permettant d'obtenir les *enfants* était toujours disponible, comme c'est le cas pour l'API *Dropbox*.

La représentation d'un fichier mis à la corbeille pose elle aussi problème, puisque le fichier n'est pas déplacé dans une corbeille « physique » mais reçoit un attribut booléen indiquant ce statut. L'API offre la possibilité de modifier l'identifiant du parent d'un fichier. On pourrait s'attendre à ce qu'un identifiant invalide ou *null* déclenche une erreur. Mais il n'en est rien ; le fichier reste existant mais n'est plus accessible au travers de *Google Drive*.

Au niveau de la création des règles, nous avons dû faire face aux potentiels conflits entre les règles créées par l'utilisateur. Pour pallier à ce problème, nous avons caché le système de prépondérance de *Drools*. Ce dernier permet de mettre un « poids » sur chaque règle pour savoir dans quel ordre elles s'appliqueront. Le problème réside lorsque deux règles ont la même prépondérance. Dans ce cas, la documentation de *Drools* précise que les deux règles s'appliqueront mais sans que nous sachions dans quel ordre. Pour éviter ce problème, nous ne laissons pas l'utilisateur renseigner la valeur de la prépondérance. La règle récupère une prépondérance automatiquement lors de sa création, selon le nombre de règles déjà

créées par l'utilisateur. Cependant, l'utilisateur peut modifier la prépondérance grâce à une interface dans laquelle il peut classer ses règles. De plus, pour éviter que deux règles entrent en conflit, nous avons aussi limité le nombre de règles à appliquer par fichier à une seule. Ceci nous permet d'éviter les conflits possibles sur un même fichier pouvant être déplacé par deux règles différentes.

f) Limitation au niveau des tests

Notre application ne présente pas une grande couverture en terme de tests. Il est, en effet, peu aisé de tester unitairement notre application puisqu'elle repose énormément sur des services externes.

Nous avons néanmoins assuré une couverture en tests unitaires couvrant les mécanismes relatifs à l'arborescence de fichiers rapatriée au sein de l'application et sur les méthodes de chiffrement.

La politique de test la plus appropriée concerne l'intégration de ces services. Cependant, nous avons préféré retarder cette partie car comme évoqué précédemment, les différentes API ont nécessité de nombreuses versions de nos propres interfaces au fur et à mesure de la compréhension des plateformes de stockage.

Une possibilité évoquée au sein de notre équipe consiste en un *mock* des services externes mais cela implique de bien les comprendre pour en reproduire le comportement, ce que nous n'étions pas en mesure de fournir avant le troisième sprint. Nous n'avons malheureusement pas pu consacrer de temps à cette tâche sur ce sprint afin de respecter le calendrier de fonctionnalités annoncées.

III. Conduite de projet

a) Premier sprint

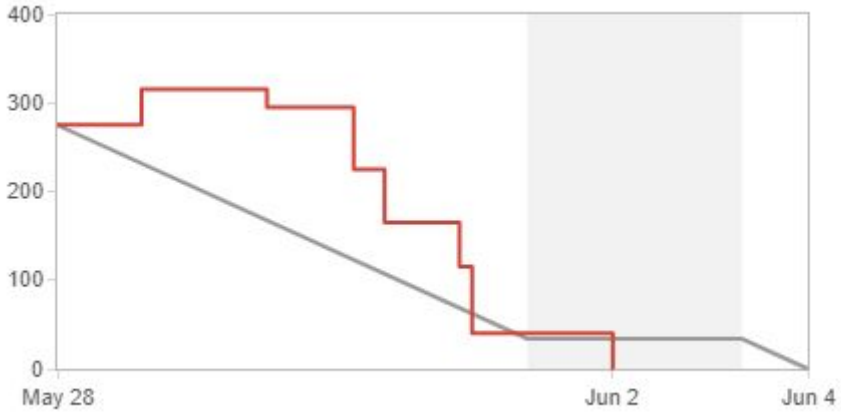


Figure n°3 : Burndown Chart de notre premier sprint

Lors du premier sprint, nous pouvons remarquer une augmentation de la valeur totale du sprint au cours de ce dernier. Ceci s'explique car nous nous sommes rendus compte en commençant à développer notre produit qu'il nous fallait ajouter une fonctionnalité permettant de visualiser notre solution sur une page Web. Si nous n'ajoutions pas ce ticket, nous ne pouvions pas présenter à notre client non-informaticien notre solution dans de bonnes conditions.

Une fois cette oubli corrigé, nous avons eu un rythme de travail satisfaisant. Malgré un début de semaine sans livraison de valeur dû à la prise en main des différents outils et API, les fonctionnalités que nous voulions implémenter se sont ajoutées au fur et à mesure et nous ont permis d'avancer étape par étape pour présenter un produit minimum viable au terme de la première semaine de développement.

b) Second sprint

Figure n°4 : Burndown Chart de notre second sprint

Dès le début du second sprint, nous nous sommes aperçus que plusieurs user stories de ce dernier se trouvaient de manière implicite dans le travail achevé à l'issue du premier sprint. Cet effet est dû à sa surévaluation, ce qui nous a incité à sortir de son cadre initial une fois les fonctionnalités validées. Ceci explique la chute importante sur notre diagramme lors du premier jour. Sur la suite du sprint, nous avons repris une progression linéaire nous permettant de livrer une version respectant notre calendrier en terme de valeur.

c) Troisième sprint

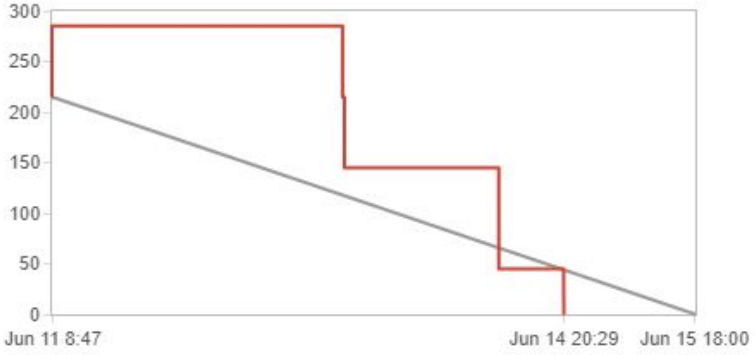


Figure n°5 : Burndown Chart de notre troisième sprint

Le troisième sprint est marqué, dès son lancement, par une hausse de sa valeur totale. Ceci est dû à l'introduction de la simulation de l'application des règles. Nous pensions au départ ne pas traiter ce ticket , mais après discussion avec notre sponsor, nous avons pris la décision de l'intégrer dans notre sprint car elle apportait une grande valeur pour l'utilisateur de notre système. De plus, il s'est avéré que nous avions surestimé le temps qu'il nous faudrait pour implémenter cette fonctionnalité, ce qui a participé à la forte baisse de la courbe du *burndown* vers le milieu du sprint. Le sprint est aussi marqué par l'introduction du support de *Dropbox*, ce qui nous a incité à débuter une phase de reprise de notre architecture afin d'y introduire l'abstraction nécessaire à l'interopérabilité. Cette partie a peu de valeur pour le client mais s'avère nécessaire pour pouvoir évoluer, d'autant plus que nous prévoyons à l'avenir d'introduire de nouveaux *Drives*.

d) Graphe radar

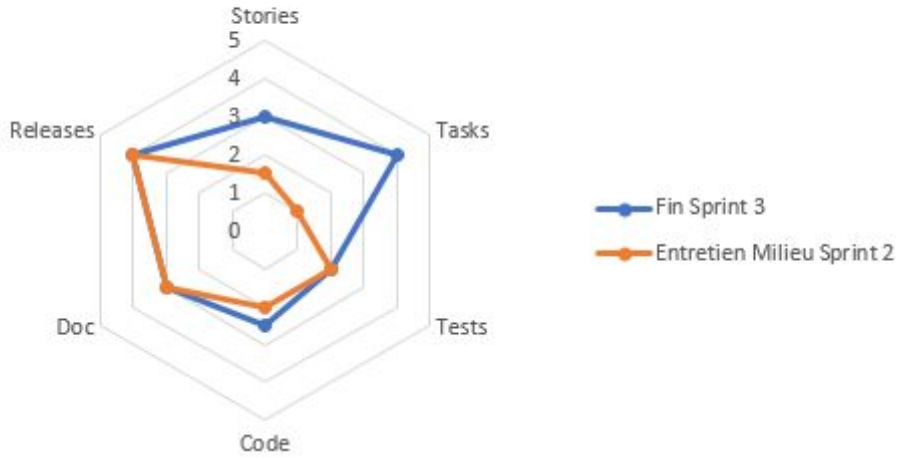


Figure n°6 : Evolution de la maturité du projet

Lors de la présentation technique, nous n'avons pas été capables de prendre assez de recul pour exprimer la maturité de notre projet. Le recul présenté s'est avéré beaucoup trop tourné vers l'aspect client du projet. Nous n'avons de ce fait pas été en mesure de montrer la pertinence de nos choix et leurs justifications. Nous sommes conscients de nos erreurs et nous les prenons en compte pour la démonstration finale de façon à présenter notre produit de manière équilibrée en ce qui concerne sa pertinence et les moyens techniques mis en oeuvre.

Nous pensons nous être principalement améliorés en terme d'analyse fonctionnelle car nous avons accordé durant ce projet une grande place aux critères d'acceptation et nous avons ciblé les fonctionnalités les plus intéressantes à développer lors des trois premiers sprints. De plus, nous avons un quatrième sprint prêt à être lancé et un *backlog* fourni en lien avec notre vision future pour notre produit. Nous avons précédemment justifié notre faible indice au niveau des tests. L'abstraction n'est pas encore complètement terminée, ce qui fait que la qualité du code n'est pas encore optimale.

Nos indices de documentation et de *release* restent constants car nous avons suivi la même logique que celle présentée durant l'entretien, le contexte actuel du projet ne

nécessitant pas de la revoir. Pour la documentation, elle reste faible car ce n'était pas une priorité absolue en terme de valeur client durant ces trois sprints. En ce qui concerne les *releases*, nous utilisons toujours une branche *master* seulement pour les livrables considérés stables, une branche *devel* pour des livraisons de fonctionnalités importantes et des branches séparées pour chaque nouvelle fonctionnalité.

e) Répartition du travail

Dans ce projet, notre objectif principal en terme de répartition du travail a été de faire en sorte que chaque membre ait pu appréhender toutes les parties du projet. Nous souhaitions avoir cette répartition pour permettre à chaque membre de l'équipe d'expérimenter les différents *framework* et API utilisés lors de ce projet et d'y apporter sa vision afin que les fonctionnalités soient les plus complètes possible.

IV. Conclusion

Pour conclure, nous sommes très satisfaits de notre projet PNSInnov. Notre produit a subi de grandes évolutions et aboutit finalement à une solution qui a séduit les deux clients que nous avons rencontrés. Ce projet nous a permis de définir nous-même un cadre et de le retravailler tout au long des 3 semaines. De plus, il nous a fallu avoir une vision plus étendue sur l'avenir qu'aurait notre projet en cas de continuation au-delà de cette période.

Par la suite, nous avons la volonté de continuer ce projet en suivant la vision à long terme. Notre backlog est déjà préparé avec un sprint prêt pour une potentielle 4ème semaine, nous sommes conscients de notre dette technique et avons pour projet de la réduire étape par étape.

Pour finir, nous souhaitons remercier l'ensemble des personnes nous ayant accompagné et guidé durant ces trois semaines de projet. Nous remercions M. Sébastien Mosser pour la pertinence de ses remarques et son aide lors de l'élaboration de notre sujet, M. Claude Galan et Mme Véronique Guérin pour leurs retours constructifs en tant que client, Mme Anne-Marie Dery-Pinna pour son soutien lors des *daily meetings*, M. Benjamin Benni pour sa critique technique nous ayant permis de nous améliorer et M. Jérôme Rancati et M. Benjamin Bourgeois pour leurs conseils. Nous souhaitons terminer par un remerciement spécial à notre sponsor M. Philippe Collet pour sa vision extérieure précieuse, son soutien sans faille et sa capacité à supporter nos jeux de mots plus ou moins bons ainsi qu'à Mme Caroline Bruno pour sa participation à notre vidéo finale.

Problème

Le rangement de fichiers est chronophage, nécessite un classement et une mise à jour permanente

Les solutions actuelles de stockage ne respectent pas la vie privée

Alternatives existantes

Classement manuel

Solution

Pilot est une solution de rangement automatique et intelligent compatible avec Google Drive, Dropbox et One Drive.
Pilot permet de chiffrer les documents exportés sur une plateforme de stockage
A plus long terme, Pilot embarquera son propre système de stockage.

Indicateurs de performance

- Volume de données transférées
- Volume de données chiffrées
- Nombre de déclenchements par heure
- Nombre de comptes utilisant la solution

Proposition de valeur unique

Une solution innovante, répondant à un besoin réel aussi bien en terme de rangement que de sécurité.

Slogan :
“Pilot, ne te laisse plus driver par le désordre."

Avantage compétitif

- S’inscrire comme LA REFERENCE dans le domaine, rester à l’écoute des utilisateurs / les fidéliser.

- Pas de concurrence actuellement, dépôt d’un brevet décrivant le concept et ses spécificités à venir.

Canaux

- Bannière publicitaire de petite taille et présence dans les journaux spécialisés

- Présence sur les réseaux sociaux, concours ponctuels permettant de gagner 3 mois d’offre classique

- Abonnement via le site

Segments de clientèle

- Les particuliers stockant des documents aussi bien administratifs que personnels comme des photos / vidéos.

- Les entreprises désirant instaurer une politique de sauvegarde organisée des données

Les particuliers et les entreprises ne souhaitant pas que l’on consulte leurs données à leur insu.

Coûts

- 4 Développeurs à plein temps sur 3 semaines : 14400€

- 2 Développeurs à plein temps (Maintenance, évolutivité): 10.000€ / mois

- 2 Serveurs dédiés en location (Offre SP-64 OVH) : 204€ / mois

- Pas de local durant la première année (Télétravail)

Total première année : 14.400 + 110.000 + 2.448 = 126.848€ (10.570€ / mois)
Années suivantes : 112.448€ (9370€ / mois ~ 3135 offres “Classique”)

Sources de revenus

- Une offre gratuite (Offre d’appel)
- Un rangement déclenché manuellement par jour ; règles standards et non personnalisables ; Limitation à un compte Google Drive

- Une offre Classique 2.99 € / mois
- Déclenchement manuel et automatique à raison de maximum 10 déclenchements par heure et par compte ; Règles personnalisables ; Limitation à un compte parmi les plateformes supportées

- Une offre Premium 4.99 € / mois
- Déclenchement manuel et automatique à raison de maximum 60 déclenchements par heure et par compte ; Règles personnalisables ; Limitation à 2 comptes parmi les plateformes supportées

- Une offre Pro 14.99€ / mois
- Déclenchement manuel et automatique à raison de maximum 60 déclenchements par heure et par compte; Règles personnalisables ; Limitation à 15 comptes parmi les plateformes supportées