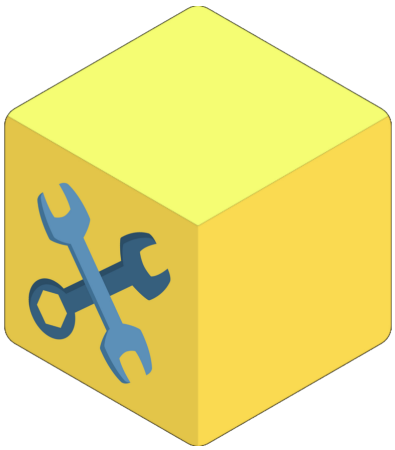
EPITA - PROJET S2



CubeTools

KM3 Team :

Kevin JAMET (Chef de Projet)

Max SAILLY

Mehdi EL ALAOUI EL AOUFOUSSI

Mateo LELONG

13.06.2022

Table des matières

1 Le projet et notre équipe

CubeTools est un explorateur de fichiers cross-plateform qui ajoute des fonctionnalités uniques et un design moderne et élégant !

Le mot d'ordre de notre projet est : ***Faciliter la vie des utilisateurs.***

Aujourd'hui, de nombreux développeurs ont recours à des machines virtuelles ou des dual boot pour le développement de leur projet. Notre équipe a remarqué une certaine frustration des utilisateurs lorsqu'il s'agissait d'envoyer et de stocker des fichiers sur des appareils n'ayant pas le même système d'exploitation. Les solutions existent mais sont souvent trop rudimentaires ou bien payantes.

Le but de notre équipe est donc de créer un utilitaire complet, un *explorateur de fichiers* avec des fonctionnalités uniques.

Notre objectif est donc à terme de répondre à la fois aux exigences du grand public, c'est-à-dire de proposer une interface agréable et facile à comprendre, mais également de répondre aux exigences des développeurs et à tout autre utilisateur souhaitant un outil performant pour la gestion et partage de fichiers, tout en proposant un accès aux services de cloud, compression et de partage de fichiers et ce de manière cross-platform.

1.1 L'histoire de notre équipe et du projet

L'équipe s'est formée naturellement du fait de notre amitié qui nous lie depuis le début de l'année 2021-2022 : Kévin, Matéo, Max et Mehdi. Nous avons décidé de faire équipe car nous avions

le même but : créer une application qui ***faciliterait la vie des utilisateurs.***.

Le nom du groupe est venu de nos initiales : le "K", et les 3 "M", d'où "KM" élevé au cube, soit "KM3" et le cube de notre logo ! (voir logo en annexe)

Nous avons tout de suite eu l'idée de faire un explorateur de fichiers à notre façon ! Car c'est un véritable outil pour tout utilisateur sur PC et toute autre plateforme. Alors pourquoi ne pas lier le nom du groupe et le terme "outil" : CubeTools !

Tout, dans le nom et les logos, est significatif pour notre équipe ! Notre lien fort pour KM3, unis sous forme d'un cube, et notre outil que nous fournissons au plus grand nombre : CubeTools !

1.2 Objectifs Atteints et Non atteints

Avant de présenter le projet plus en détail, nous allons nous attarder sur les objectifs que devait atteindre notre groupe pour cette soutenance et ce que nous avons fait, faisons le point sur : le 'à faire' et le 'fait'.

1.2.1 Reprise du Cahier des charges

Dans cette partie, nous allons reprendre en détail les éléments principaux que devait contenir notre projet.

Les 3 objectifs que nous nous étions fixés sont les suivants : Créer un explorateur de fichier fonctionnel, Proposer un transfert de fichier vers des clouds et en FTP, Créer une solution cross-plateform.

Voila les objectifs détaillés de notre projet :

— Un utilitaire de type explorateur de fichiers

— Un logiciel cross-plateform : Windows, MacOS, Linux

— Une interface utilisateur simple avec de la personnalisation
— Un envoi de fichier en FTP
— Un envoi de fichier avec l'API Google Drive
— Un envoi de fichier avec l'API One Drive
— Une intégration de la librarie de compression 7z (BONUS)
— Une intégration complète de Smash et Snap Drop (BONUS)
— Un utilitaire bluetooth 'NearBySend' pour l'envoi de fichier via bluetooth (BONUS)

Voici maintenant un tableau résumant l'avancement théorique que notre groupe avait proposé lors de la création du cahier des charges (voir le tableau théorique de l'avancement)

1.2.2 Notre résultat

Et voilà le résultat : le contrat est plus que rempli ! Tous les objectifs (hors bonus) ont été implementés et sont fonctionnels, l'intégration de Smash et Snap Drop sont partiels et la compression est complétement fonctionnelle. Par ailleurs, nous avons voulu rendre notre logiciel très personnalisé, le système de personnalisation est très complet.

Les membres de l'équipe KM3 ont su s'organiser tout au long du projet pour être aussi efficace que possible. Nous avons divisé les tâches par catégorie. Chacun s'est vu attribué des tâches pour lesquelles nous étions motivés. Cette répartition a été définie très tôt afin d'anticiper au maximum les problèmes que nous allions rencontrer.

Commençons par le travail de Mehdi. Il s'est chargé de la partie One Drive, donc de l'implémentation des APIs et de l'organisation des fonctions en Back-End pour lier le code au UI. Ses tâches ne se résument pas uniquement à cela, il a mis en place les APIs

de compression/décompression avec 7zip, l'intégration du FTP et du OneDrive dans le UI. Il a également participé à la liaison du code AXAML avec Matéo et la création des clients modulaires pour l'implementation abstraite des connexions clouds et de l'interface locale.

Max, quant à lui s'est chargé du WebSite, d'une partie du UI et de l'API GoogleDrive. Il a été très investi pour réaliser ces tâches, notamment le website. Il a fait entièrement le WebSite et l'intégration de GoogleDrive dans le UI ainsi que les fonctions backend de ce cloud. A l'aide de Matéo et Mehdi, Max a pu implémenter de façon modulaire l'API Google Drive dans le code-behind de notre UI.

Matéo s'est chargé de la partie Back-End de notre logiciel, c'est à dire la partie C#, les fonctions et le code behind pour le CLI et par conséquent le UI, sans oublier leur intégration dans le UI. Il s'est occupé de choisir le modèle de structure du projet (MVVM puis MVC), de coder les fonctions nécessaires pour le bon fonctionnement de l'application et notamment la partie locale. Il avait la lourde tâche avec Mehdi d'adapter le Backend pour faire fonctionner l'application avec les libraries. L'équipe s'est fortement mobilisée sur l'implémentation car c'est sur ce code que reposait une bonne partie de notre application.

Enfin, le chef de projet, Kévin, s'est occupé de tout ce que l'utilisateur peut voir, à savoir le UI, les popups, les icônes. En tant que chef de groupe, il a également soutenu l'organisation du groupe pour les soutenances et les oraux de soutenance. Son rôle a été primordial pour remplir notre objectif : obtenir un UI plaisant, simple et personnalisable.

1.3 Un Logiciel Cross-Plateforme avec Avalonia

Dans cette partie, nous allons principalement parler des frameworks utilisées pour créer notre utilitaire.

Pour rappel, plusieurs solutions avaient été longuement étudié auparavant afin de ne pas devoir recommencer le projet.

Au début, nous avions pensé à utiliser la technologie WPF proposé par Microsoft. C'est une framework très pratique pour créer un logiciel comme le nôtre mais, hélas, qui n'est disponnible que sur Windows. (voir WPF)

Nous avions ensuite pensé à créer notre interface graphique en utilisant la librarie GTK. Cette librairie est très puissante puisqu'elle est développée à partir du projet MONO qui est cross plateforme et très documenté mais le manque de compatibilité avec un IDE et surtout les problèmes de dépendances avec les DLL générées nous ont fait changer d'avis. (voir GTK)

Nous nous sommes ensuite tournés vers les solutions adaptées de Microsoft. Nous nous sommes tournés vers l'ASP.NET puisque l'interface graphique peut être créée avec du code HTML et CSS. Cette solution n'était malheurseument pas pertinente puisque l'ASP.NET est orientée évènement et n'est pas adaptée pour un explorateur de fichier local.

Pour ce projet, nous avons finalement choisi la framework Avalonia car après de nombreuses recherches, cette framework est celle qui répond au mieux à nos besoins car elle est à la fois cross-plateform et proche du XAML utilisé par le WPF. Nous avons utilisé la version .NET 6 de Microsoft qui propose un grand nombre de fonctionnalités déjà cross-plateform.

1.4 Concurrence

Dans cette partie nous allons reprendre en partie les autres solutions qui existent et qui peuvent être considérées comme des concurrentes.

Files est un concurrent très puissant car il est vraiment optimisé pour Windows et propose toutes les fonctionnalités basiques d'un explorateur de fichiers tel que la gestion de disques, dossiers, fichiers en local, mais également en distant avec les disques réseau et le support de solutions cloud. Cette application est également très optimisée pour les ordinateurs portables de par son optimisation et son faible coût en mémoire et en ressources processeur. Il propose également une intégration bien meilleure de OneDrive et GoogleDrive que la nôtre. Cependant, il reste disponnible uniquement sur Windows. (voir FILES)

CX File Explorer et Solid Explorer sont également des concurrents de CubeTools de par leurs fonctionnalités. En effet, ils intègrent de nombreuses fonctionnalités comparables à celles de Files décrit précédemment mais sur Android. Ce qui en font des outils très puissants proposant de nombreuses fonctionnalités et bien plus que les besoins d'un utilisateur sur Android. (voir CX screen)

1.5 Notre solution unique

Les solutions précédentes, malgré leur optimisation et le nombre de fonctionnalités disponibles, ne sont pas comparables à notre solution.

Par exemple, Files a été développé pour Windows et CX File Explorer et Solid Explorer sont des applications uniquement disponibles sur Android. Certes, elles proposent de nombreuses fonctionnalités mais ne sont pas aussi simples d'accès que Cu-

beTools

Ce qui rend CubeTools unique est également sa framework : Avalonia. (voir icone d'avalonia)

Comme décrit précédemment, cette framework permet de rendre notre application Windows cross-plateform et compatible également avec Linux et MacOS.

L'équipe KM3 voulait absolument développer l'accès aux fichiers simple et fluide entre les appareils, même pour un débutant !

Nous avons donc opté pour des sites de partage de fichiers tels que Snapdrop et Smash pour partager ses fichiers sur un même réseau ou par mail, ou tout autre plateforme de communication et ce de manière fluide, rapide, simple et surtout gratuite !

Ces sont pour ces mêmes raisons que notre application est unique et qu'elle peut être utilisée par le grand public ; Une application simple, moderne et proposant toutes les fonctionnalités essentielles pour améliorer le workflow de l'utilisateur ; Et pour les nostalgiques, CubeTools existe également en CLI pour l'utiliser en lignes de commande !

2 Les bases du projet : 1ère période

Dans cette partie, les bases du projet et la façon dont nous avons procédé pour proposer notre premier Minimum Viable Product (MVP) vont être abordés.

2.1 Le CLI ou Command Line Interface

Dès le début de notre projet, il y avait de nombreuses choses que nous ne savions pas faire.

La version CLI (command line interface) de notre projet est apparue comme une solution évidente au vu de notre manque de connaissance pour la création du UI. Nous avons dû passer du temps et se donner les moyens pour réaliser l'interface utilisateur.

Une version plus simple à implementer pour créer notre premiere librairie et nos classes pointeurs, permet par ailleurs de laisser plus de temps pour débugger la solution.

La première version du CLI reposait sur la lecture de string en direct. Nous ne voulions pas nous attarder à faire un système de tokenization d'une chaine de caractères. Le CLI reste fonctionel et est surtout une version de test pour la librarie du local. (voir screenshot du CLI)

Concernant les fonctionnalités présentes, on retrouve les fonctions basiques à savoir :

— cp : Copie un fichier ou un dossier et lui donne un nouveau nom dans le dossier chargé.
— rm : Supprime de façon définitive un fichier. Si c'est un dossier, il le supprime de façon définitive et recursivement.
— rmdir : Supprime de façon récursive le dossier passé en paramètre

— ls : Affiche tous les dossiers du dossier actuellement chargé, affiche également des propriétés comme la taille ou la date de modification.

— mv : Renomme le dossier (en récursif) ou un fichier avec le nom donné.

— cd : Change le dossier courant avec un chemin relatif.

— cat : Affiche le contenu du fichier donné en paramètre, renvoie une erreur s'il s'agit d'un dossier

— touch : Créer un fichier vide

— mkdir : Créer un dossier vide

— search : Suivie d'un pattern, il permet de chercher de facon récursive un pattern de type regex dans le dossier courant

— open : Permet d'ouvrir avec une application proposée par défaut par l'OS le fichier donné en paramètre.

2.2 La librarie Locale : BackEnd

La création du Backend était l'opportunité pour apprendre comment marche un explorateur de fichiers.

En effet, aussi simple que semble son fonctionnement, il n'est pas aisé de développer un file explorer, cela ne s'est pas fait du jour au lendemain. Il a fallu penser le fonctionnement et ne pas se jeter sur la création de fonctions inutiles.

La première étape pour la création des fonctions en backend a été de penser le fonctionnement d'un File Explorer et de prendre les fonctionnalités à notre portée. Le principe est le suivant : on a un dossier chargé et ses fichiers / dossiers affichés, on peut naviguer, on peut apporter des modifications comme copier un fichier, détruire un dossier. Un système d'évenements doit etre mis en place afin de mieux refléter la dynamique du client local ou distant.

La seconde étape a été de préparer des classes adaptées pour résoudre les différents problèmes rencontrés. Nous avons rencontré des problèmes de mémoire importants comme le problème pour stocker les enfants (sous - fichier/dossier) d'un dossier. Il ne fallait surtout pas stocker tous les enfants de manière récursive. De la même facon, il fallait avoir une référence vers le parent.

La troisième et dernière étape a été de créer ces fonctions de façon optimisée :

- Une première version de la librarie locale avait été proposée. Cette dernière a vite été remplacée après la 1er soutenance puisqu'elle n'était pas assez complète et pas adaptée.

- Une seconde version de la librarie complète a été proposée mais ne pouvait s'adapter aux clients modulaires permettant d'intéragir avec les clouds distants.

- La dernière version proposée à cette soutenance a été optimisée à son maximum et est complète.

2.3 Premier Système d'erreur : ManagerException

Un système d'erreur a été implementé dès la création du CLI. Pourquoi créer ce système ? Tout simplement parce qu'il est pratique à traiter dans le UI et permet facilement d'informer.

Le système fonctionne sur le principe des couches de notre logiciel. La partie bas niveau, traitée par les fonctions microsoft génèrent des erreurs, on vient récupérer les erreurs dans le code haut niveau. Ces erreurs sont converties en erreur plus compréhensible pour l'utilisateur et on les affiche à l'aide de popup dans le UI.

La génération de popup se fait de façon modulaire, c'est-à-dire, chaque popup d'erreur a la meme base mais les fonctonnalités apportées sont différentes.

Dans cette première période, l'implementation du UI n'était pas commençée donc il était impossible de gérer la seconde partie du système, nous avons juste opéré à la transformation des erreurs bas niveau en erreur haut niveau. (voir CLI Errors)

2.4 Le UI

Le UI n'était pas notre priorité pour cette première période car nous nous sommes fixés de faire les fonctions nécessaires pour le terminer pendant la deuxième période. Cette première période faisait office de phase de découverte et de test pour découvrir les possibilités du AXAML.

2.4.1 Le langage de balisage utilisé : AXAML

Le plus compliqué pour notre équipe était de s'adapter à la framework Avalonia puisque celle-ci se base sur les principes du WPF (Windows Presentation Foundation) qui est une framework UI pour réaliser des applications de bureau. Ce mode de programmation avec une interface graphique est très différent de la création d'algorithme ou de programme en ligne de commande.

Concernant la framework en elle-même, Avalonia reste proche du WPF malgré quelques fonctionnalités manquantes. Elle est moins documentée et manque souvent de fonctionnalités récentes proposé par le WPF. Certains attributs peuvent différer et Avalonia ne propose pas forcément les mêmes fonctionnalités que le WPF, ce qui rend la tâche encore plus complexe. Il est parfois même nécessaire de refaire toute l'implémentation uniquement parce que'une fonctionnalité est manquante avec la framework que nous utilisons.

Le langage utilisé pour créer l'interface graphique est le AXAML dérivé du XAML. C'est un langage de balisage utilisé pour la

partie graphique, c'est la version Avalonia du XAML. Le XAML ressemble au HTML dans sa structure car c'est aussi un langage de balisage et est donc assez intuitif à utiliser. Par exemple, il suffit pour mettre un bouton de créer une balise bouton et tout ce qui se trouvera entre les balises du bouton appliquera les modifications uniquement sur le bouton.

Le XAML (ou AXAML) est un langage de balisage très puissant lorsqu'il est associé à un fichier C# : c'est ce qu'on appelle le code-behind. Dans ce fichier, il est possible d'utiliser toutes les fonctionnalités du C# mais on peut également faire des liens avec des éléments du UI. On peut par exemple prendre la références d'un élement du UI, ajouter des événements, modifier la couleur du controller, fermer une fenêtre...

Il est également possible de créer des styles. Si un style est créé avec le nom "ButtonStyle1", et que certains boutons sont initialisés avec ce style défini, ils appliqueront tous les paramètres sur celui-ci. Le XAML permet donc à la fois d'appliquer du style mais aussi de rendre fonctionnelle une interface statique avec le code-behind.

Il est donc relativement simple de faire une petite application avec ce langage car il ressemble au HTML et CSS. Avec Avalonia, les options sont relativement limitées et les fonctions assez compliquées à utiliser mais elles sont suffisantes pour permettre de développer CubeTools.

2.4.2 Construction de la première interface

Nous avons donc appris le AXAML pendant quelques semaines. Nous étions donc en mesure de faire une application (uniquement la partie visuelle, sans lisaison avec le code C#).

Nous avons opté pour une première version du UI en bleue (présente dans les annexes). Cet UI était provisoire et donne un

aperçu de notre logiciel.

Pour créer ce 1er UI, nous avions utilisé les bindings en nous inspirant du principe du MVVM. Les bindings sont des getters et setters qui lient le code C# et le AXAML.

Le principe des bindings est de laisser le code se charger du rafraichissement du UI à l'aide de notification. Les bindings nous ont posé beaucoup de problèmes parce qu'ils ne sont pas vraiment utiles pour le développement d'une application comme CubeTools. Il s'agit bien évidemment de la solution la plus adaptée mais nous nous sommes rendus compte qu'il était également possible de manipuler le UI uniquement avec le code-behind. Dans cette première période, nous avons passé beaucoup de temps à essayer d'adapter le code à ces bindings.

Cette implémentation ne marchait pas vraiment et était très instable, lorsque des boutons étaient cliqués, des plantages étaient très fréquents. Nous avons donc décidé de continuer le UI sans s'occuper de la liaison avec le code.

Nous avons donc développé notre première version du UI qui était peu design et très rudimentaire en terme de fonctionnalités (à cause de notre implémentation avec les bindings).

2.4.3 Des problèmes de performance

Parlons mémoire. La première version proposait un UI avec un coût mémoire de 800MB. De plus nous avions mis toutes les barres en taille dynamique, ce qui demande beaucoup de ressources en mémoire et en calculs. Le problème de mémoire venait principalement du fait que les icones chargées avaient une trop bonne qualité pour le rendu, de même pour le nombre de pointeur qui était chargé de manière récurisve. On venait littéralement de charger un disque dur en mémoire sous forme de pointeur. (voir avancement de l'optimisation)

Concernant l'interface en elle-même, elle n'était pas aux couleurs et au design de CubeTools c'est à dire dans des couleurs chaudes, un design moderne et élégant. Ici, nous avions juste développé un UI pour faire nos tests et nous familiariser avec le langage AXAML avec des icônes téléchéchargées.

Pour finir, l'interface avait des rendus statiques ce qui rendait impossible le fait de redimenssionner.

Pour la seconde période, l'UI allait être notre priorité pour apporter une version pratiquement 100% fonctionnelle et aux couleurs de CubeTools !

2.5 Le site web

Pour cette première période, KM3 a décidé de mettre en place un site Web fonctionnel et performant. C'est Max qui s'est occupé de cette partie.

2.5.1 Les débuts

Dès le début, Nous avons commencé à travailler sur le site Web, et Max n'ayant jamais fait de développement web avant, il s'est porté garant pour réaliser le site web de notre projet.

Il fallut tout d'abord se documenter sur le sujet grâce aux nombreux tutos que l'on peut trouver sur internet. En parallèle, Max commencait déjà à imaginer la structure et les différentes informations qu'il allait mettre sur le site web.

2.5.2 Premières avancées

Les tâches que Max devait réaliser dans le projet en lui-même nécessitaient une avancée conséquente sur certains points importants notamment dans la partie du UI. Ainsi, Max a réalisé une grande partie du site web avant même la première soutenance.

Nous avions comme idée de réaliser une seule page web contenant toutes les informations nécessaires aux utilisateurs pour utiliser facilement notre explorateur de fichier. Max a dans un premier temps créé, la barre de navigation permettant à l'utilisateur d'accéder aux différents parties de la page web, ainsi qu'aux liens que l'on jugait important tel que notre repos Git. Il a aussi réalisé de nombreuses animations pour rendre le site web plus attractif et professionel, en cohérence avec notre projet initial.

2.5.3 Les difficultés rencontrées

N'ayant jamais fait de dévelopement web, il y a quelques notions que Max a eu du mal à comprendre, notamment en javascript ou en css lorsqu'il s'agit de positionner de la bonne manière un objet dans l'espace. Le javascript était aussi un language nouveau, ainsi les premières animations étaient assez basique, et le javascript très peu utilisé.

2.6 APIs

2.6.1 l'API de Compression

Parmi la quantité de fonctionnalités que nous voulions ajouter à CubeTools, le support de la compression/décompression nous a paru évident et necessaire. Mehdi s'est occupé d'implémenter la librarie de 7z dans la partie backend du projet.

Il fallait tout d'abord faire un choix pour les algorithmes de compression à supporter. D'après nos expériences personnelles et des recherches sur internet, nous avons donc choisi de supporter le ZIP, algorithme de compression le plus répandu, et le LZMA (7zip), algorithme de compression très efficace et aussi connu.

Par la suite, nous avions dû choisir l'implémentation de ces algorithmes de compression et de décompression à CubeTools. Il était certes possible de comprendre comment marchent ces algorithmes et les refaire à la main, ça allait prendre beaucoup trop de temps et ce n'était pas notre objectif principal. Il fallait donc opter pour l'utilisation d'une API.

Au départ, nous avions opté par un ensemble de fonctions déjà implémentées en .NET Core 6.0 pour l'algorithme de ZIP, et l'API SevenZipSharp, qui lui même reprend l'api de SevenZip pour l'algorithme du LZMA. Nous nous sommes rendus compte plus tard que SevenZipSharp supportait aussi le ZIP. Pour uti-

liser ces algorithmes de compression, nous avons utilisé le dll de la librarie de 7z necessaire pour utiliser l'API de SevenZipSharp.

Pendant le processus d'implémentation de cette API il était nécessaire d'utiliser les classes que Matéo avait implémenté dans son API backend de gestion de fichiers afin d'avoir une représentation objet des différents fichiers/dossiers à compresser.

L'avantage de cette API, c'est qu'il est possible de personnaliser et de paramétrer ce qui se passe pendant le processus de compression ou de décompression. Par exemple, nous avons ajouté une option de multithreading qui permet d'accélérer considérablement ce processus. Par la suite nous avons ajouté des events afin de suivre où en est ce processus et quand il se finit.

2.6.2 Difficultés rencontrées pour la compression

Le principal souci pour la compression a été de trouver une librarie qui supporte le LZMA. Le LZMA a l'avantage d'être très efficace et surtout d'être cross-plateform grâce à une dll portable.

On arrive donc au deuxième problème, charger le dll dans le code C# afin d'utiliser la bibliothèque de compression. Une des seules API utilisant un dll capable d'être utilisé facilement en C#

2.6.3 L'API du File Transfer Protocol

Lors de cette première période, Mehdi s'est occupé de la création du système permettant de naviguer et d'appliquer des opérations aux fichiers d'un serveur FTP.

Durant cette première période, il a utilisé l'API proposée par FluentFTP compatible avec C#. Il s'est occupé d'implémenter les fonctions permettant de récupérer les sous-fichiers et sous-dossiers d'un dossier, création, destruction d'un dossier et d'un fichier ...

La création de classe permettant de pointer vers un fichier ou dossier a été necessaire pour l'implémentation dans le UI, Mehdi s'est donc chargé de créer un système de pointeur permettant de mieux naviguer dans notre interface.

Concernant l'implementation du FTP dans le UI lors de cette période, elle n'a pas été réalisée car elle ne faisait pas partie des objectifs de cette période.

2.6.4 Difficultés rencontrées pour le FluentFTP

Les problèmes rencontrés lors de l'implémentation du FTP ont été nombreux. Mehdi a remarqué que des fonctionnalités étaient

inexistantes comme par exemple le SFTP et la récupération de propriétés sur des fichiers ou des dossiers (taille, date de modification ...)

Lors de la deuxième période, Mehdi est donc passé à une nouvelle implémentation en passant directement par des requêtes http

3 Nouvelles implémentations et UI : 2ème période

Lors de cette deuxième période le but était de fournir une application quasimment 100% fonctionnelle avec les fonctions basiques d'un explorateur. Nous avons donc refait entièrement l'implémentation plusieurs fois, refait l'UI plusieures fois pour avoir une application qui approche le plus de sa version finale.

Le réel défi de cette deuxième période a été de rendre dynamique notre interface utilisateur et donc de relier les libraries déjà créées.

3.1 Librarie Locale : Backend

Lors de cette seconde période, la pluspart des fonctionnalités de la librarie locale qui se chargeaient de générer des pointeurs et d'y appliquer des opérations étaient déjà mis en place, il fallait cependant adapter le code pour le nouvel UI.

Matéo s'est donc chargé de passer en Threadé la plupart des fonctionnalités coûteuses en opérations et longues à effectuer à savoir la Copie, la Destruction et le fait de Renommer.

Cela n'était pas chose facile car l'actualisation de l'interface devait se faire en Threadé et nécessitait un Thread spécial proposé par Avalonia 'Dispatcher.UIThread'.

L'utilisation des classes de la bibliothèque du F# (FileSystem) permet d'utiliser les fenêtres proposées par Microsoft dans son file explorer

Par ailleurs, chaque fonction a dû être partiellement réécrite pour récupérer les exceptions bas niveau et les transformer en exception haut niveau.

3.2 Système de popup d'erreur

Le système de popup d'erreur a été principalement implémenté lors de cette deuxième période.

Le principe de ce système repose sur le principe que chaque Exception Haut niveau peut etre rattrapé dans le code-behind lors de l'appel aux fonctions des libraries. L'exception haut niveau est interprétée par la popup qui vient se charger et s'afficher sur l'écran. (voir Error Popup)

Le travail a été pharaonique car il fallait, pour chaque appel vers une fonction de la librarie pouvant potentiellement engendrer une erreur, générer une popup d'erreur valide et réactive.

3.3 Nouvelle implémentation

L'interface utilisateur a necessité l'utilisation d'une implémentation particulière afin de rendre le UI dynmaique et synchronisée avec le backend.

Qu'est qu'une implémentation ? C'est une façon de coder l'interface graphique, c'est-à-dire, la façon dont on vient faire la connexion entre l'interface graphique et le code-behind en C#.

Il existe de nombreuses implémentations. Les deux principales implémentations pour l'interface utilisateur sont le MVVM (nouvelle implémentation reposant sur le binding) et le MVC (ancêtre du MVVM). C'est ces dernières sur lesquelles nous nous sommes reposés tout au long du projet.

3.3.1 Passage au MVVM

Le MVVM est un modèle qui se base sur le système de bindings. Un binding est un getter/setter permettant de modifier et de rafraichir l'interface en direct. On utilise le principe du reactiveUI

qui vient notifier d'un changement de valeur dans le code-behind et vient le rapporter dans l'interface utilisateur.

Le modèle MVVM est un nouveau modèle qui s'appuie sur 3 composantes (les couches du modèle) : le modèle, la view et la view-model. (voir schéma du MVVM)

Chaque composant a son propre rôle :

— View : L'affichage de l'interface utilisateur qui vient notifier des changements auprès du ViewModel à l'aide de bindings directement dans le code axaml.

— ViewModel : Il opère à la liaison entre le Model et la View. Il notifie des changements de la View au Model et inversemment.

— Model : Le model est la partie fonctionnelle de l'application. Elle stocke les variables necessaires pour l'interface utilisateur et plus important encore, les fonctionnalités du BackEnd.

3.3.2 Quelle modèle pour notre projet ?

Une réflexion autour du modèle à utiliser pour notre projet est apparue lorsque nous avons commencé à implémeneter le UI.

Voici le problème rencontré lors de l'implémentation du MVVM : Il était très compliqué d'éditer en tant réel les variables, il y avait toujours une désynchronisation entre ce que l'on voyait et les variables en arrière-plan.

Par ailleurs, nous avons déjà créé en partie des libraries du BackEnd de notre logiciel, il nous a paru compliqué d'utiliser le modèle MVVM. Ce modèle est particulièrement performant pour des logiciels dont le code-behind est très relié au code AXAML.

Le modèle MVVM n'est donc pas optimisé pour notre projet ne serait-ce que par son caractère objet et sa complexité de mise en place.

3.3.3 Le MVC

Nous avons donc opté pour le MVC (Model View Controller).

Le principe est clair : Le Controller "contrôle" la View et vient l'éditer en temps réel. A l'inverse, la View est éditée par le controller mais permet également d'éditer les variables des controllers via des events.

Et donc voici le principe du MVC avec notre implémentation et Avalonia :

Chaque View possède son propre code-behind qui hérite de la classe associée à la vue. La View envoie des events qui peuvent s'accrocher sur différents composants de notre View (un boutton, un panel ...). Cet event est appelé dans le code-behind associé et envoie par ailleurs deux paramètres (voire plus) qui sont en général le 'sender', c'est-à-dire, l'objet qui a levé l'event et le

type d'event. Cela peut être un évènement de type 'PointerPressed' qui détecte un click de souris ou bien encore un événement de type 'KeyPressed' qui détecte si une touche est pressée.

A l'inverse, le code-behind peut stocker des références vers des éléments du UI dans des variables à l'aide d'un attribut 'Name' et de la fonction 'FindControl<TYPE>(NAME)' qui vient trouver l'élément de type TYPE et ayant un nom NAME

C'est ce modèle que nous avons adopté pour notre interface utilisateur. (voir Notre modèle)

3.4 Le UI

Pour cette deuxième période, nous avons réalisé plusieurs UI qui nous ont permis de comprendre le code et donc d'implémenter les différentes fonctionnalités d'un explorateur de fichier, comme copier, coller par exemple. Les premières versions étaient peu esthétiques et peu travaillées telle que celle présentée à la première soutenance. (voir CubeTools Soutenance 2)

3.4.1 Développement

Nous avons donc commencé le développement d'un nouvel UI, car le précédent ne nous convenait pas. Nous sommes repartis de la première version pour l'optimiser peu à peu et arriver à un design plus satisfaisant. Dans cette version (absente des annexes), nous avons juste modifié certaines icônes et changé la couleur de l'interface pour du blanc. L'interface était fade et peu agréable à utiliser.

Mais nous l'avons gardé quelques semaines pour tester différentes implémentations. Une seule a marqué l'équipe : le MVC (décrit précédemment).

De façon concrète, le MVC nous permet de lier une variable référencée dans le UI avec une variable directement dans le code C# qui peut être traitée en Back-End. Par exemple, il est possible de lier une TextBox avec une variable, que l'on nommera "LaBoiteDeTexte". Dans le code C#, on pourra récupérer la variable dans cette TextBox pour la traiter dans le code-behind, c'est à dire dans nos fonctions qui traitent le texte pour renommer un fichier par exemple.

3.4.2 L'interface de la 2e période

Une fois cette structure adoptée, il nous est possible de tout faire ! Kévin a donc refait tout le UI de A à Z, en repartant

de 0. Il a commencé par établir une palette de couleurs qui se rapprochent le plus possible de l'icône mais également des tendances actuelles, telles que des couleurs vives et dégradés, effets sur les icônes.

Il a décidé de partir sur une palette de couleurs ciblée sur le jaune et le orange pour raviver l'interface trop terne réalisée auparavant.

Pour le design, Kévin a décidé de suivre le design du site web, c'est à dire, un site minimaliste et vintage car la combinaison des deux est très tendance ces dernières années !

Pour les icônes tout a été refait à la main par Kévin car jusqu'ici, aucun pack d'icône ne nous plaisait vraiment. L'équipe a alors décidé de réaliser toutes ses ressources elle-même ! Plusieurs versions (3 au total) ont vu le jour. La première était trop "standard" car elle ressemblait énormément à ceux de Windows. La deuxième était trop colorée et pas dans les teintes de couleurs de l'application.

La troisième version, plus originale est simple et moderne : des formes géométriques simples avec le moins de détails possibles, plutôt rectangulaire avec des bords arrondis et une ombre orangée pour rappeler les couleurs de l'application. Le but étant d'avoir une icône simple, moderne, minimaliste et surtout explicite pour ne pas perdre l'utilisateur. Le but d'une icône reste la sobriété tout en gardant l'information d'un message textuel sous forme d'image, c'est le défi de les réaliser soi-même avec PhotoEditor et Background Eraser sur Android (dans les annexes) !

Après quelques jours tous les icônes étaient fin prêtes pour être intégrées dans le nouveau UI ! Kévin a décidé de tout recommencer en repartant sur de bases solides avec des brouillons pour être sûr de la version finale de l'application. Cela passe par des comparaisons entre logiciels concurrents, tendances ac-

tuelles pour les designs d'applications. De nombreux brouillons ont été réalisés puis le UI programmé en repartant sur une page blanche.

Le nouveau UI flambant neuf est enfin prêt a être implémenté, aux couleurs de CubeTools, moderne et bien plus ergonomique que les précédentes versions (voir annexes).

Une fois notre nouveau UI programmé, nous avons commencé à implémenter les fonctions en Back-End dans notre UI. Mais un problème s'est alors posé : les performances ! CubeTools était devenu TRES volumineux en mémoire et en ressources sur le processeur en ayant enlevé des éléments comparé au précédent UI, pour le rendre plus minimaliste. Le problème est expliqué dans la rubrique suivante.

Nous avons maintenant une application qui ressemble grandement à sa version finale (dans son design et le UI) ! Avant même de faire l'implémentation, Kévin a également réalisé les popups car toute la partie affichage est sa tâche tout le long du projet !

Les popups doivent être modernes et simples pour, encore une fois, ne pas perdre l'utilisateur ! C'est pour cela que l'icône de l'action à réaliser sur le fichier/dossier est affichée sur la partie gauche en gros avec les zones de textes ou choix possibles sur la droite. Cela rend l'application beaucoup plus aérée et simple d'utilisation, le tout avec un design qui donne envie d'utiliser l'application ! (voir annexes)

La partie Ui était fin prête. Il fallait maintenant avoir une implémentation fonctionnelle pour cette fin de deuxième période.

3.4.3 Implémentation du Code-Behind

Nous avons donc décidé, avec notre toute nouvelle application flambant neuve, de terminer au moins la partie locale avec les

fonctionnalités de base d'un explorateur de fichiers.

Nous avons donc commencé à implémenter les fonctions mais un autre problème s'est posé : notre implémentation ne suivait pas tout à fait le modèle du MVC. Nous avons donc dû réadapter notre code pour le rendre fonctionnel.

Nous avons donc commencé à séparer la définition des vues de notre interface en sous-vue afin de rendre l'interface modulaire et réutilisable.

Pour cela, Matéo et Kévin ont cherché, avec l'aide de Mehdi à faire cette nouvelle implémentation et intégration dans le UI.

Nous avons procédé à la séparation du UI en sous vue de la façon suivante : une zone permettant d'intéragir avec des liens statiques (LinkBar), une zone permettant de naviguer (NavigationBar), une zone permettant réaliser des opérations sur les pointeurs (ActionBar) et enfin une zone affichant les pointeurs actuellement chargés (PointersView)

L'équipe avait enfin une implémentation fonctionnelle, il fallait juste raccorder les fonctions avec la méthode du MVC avec le UI. Rien de plus simple car tout dans le Ui était préparé ! Il fallait juste appeler les fonctions en Back-End avec le ".cs" du fichier AXAML avec lequel il est lié, comme avec n'importe quelle classe C# .

Pour cette fin de deuxième période nous avons donc trouvé une implémentation adaptée pour présenter ici, une application fonctionnelle !

3.4.4 Un problème d'icone

Lors de la première période, nous avions rencontré des problèmes importants rendant l'utilisation de notre application impossible

Après de très nombreuses recherches, l'équipe s'est rendu compte

que ce coût mémoire venait en réalité des icônes, car Kévin les avait réalisées en 2160x2160 pixels, ce qui est bien trop volumineux pour une application !

Il a donc cherché des informations sur les tendances actuelles et voir jusqu'à où les icônes peuvent être compressées sans aucune perte de qualité perceptible à l'oeil humain. Après de nombreuses recherches, Kévin a déccouvert que le standard était le 256x256 pixels pour les petites icônes et 512x512 pour les icônes d'application.

Maintenant il fallait chercher sur quels écrans les utilisateurs peuvent utiliser notre application. Le standard est le 1920x1080 pixels mais de plus en plus d'utilisateurs (notamment professionnels ou gamers) achètent des écrans de bien meilleure résolution telle que la 2K (QHD : 2560x1440) ou la 4K (UHD : 3840x2160). Kévin a alors décidé de mettre les icônes en 270x270 (soit un petit peu au dessus de la norme) car les écrans peuvent afficher, avec les modèles les plus onéreux, entre 200 et 300 pixels par pouce, et jusqu'à 500 pour les petits écrans 4K en 17 pouces par exemple, sur lequel nous avons fait tous nos tests.

Après quelques calculs, le 270x270 était parfait car on divise la résolution par 8 tout en divisant la taille du fichier par 10 à 20 ! De plus, les icônes font moins d'un pouce d'envergure, donc même si l'écran permet d'afficher du 500 pixels par pouce, il sera impossible de voir la différence !

Avec cette optimisation nous avons réussi à diviser par 3 à 4 les coûts mémoire ! Notre application était de nouveau fluide pour atteindre un niveau de fluidité que l'équipe n'avait jamais eu avec CubeTools !

3.4.5 Un problème de pointeur

Du côté du code, nous avons réduit le coût mémoire imposé par la génération des pointeurs en faisant hériter les pointeurs de la classe IDisposable

L'optimisation s'est faite sur deux niveaux car à la fois les pointeurs ne devaient pas être générés de facon recursive afin de ne pas charger toute la mémoire de l'ordinateur et à la fois nous devions détruire ces pointeurs pour éviter des fuites de mémoire.

Le fait de rendre les pointeurs 'Disposable' permet de les effacer "à la main" de la mémoire du programme. Evidemment, le Garbage collector s'occupe déjà de le faire mais lorsqu'il s'agit d'un très grand nombre de pointeurs, le Garbage collector ne s'avère pas suffisament réactif pour détruire les pointeurs dès qu'ils deviennent inaccessibles.

3.5 Website

Le site web étant déjà bien avancé pour la soutenance à venir, Max ne s'est pas focalisé sur son avancée, il a notamment continué à mettre à jour les tâches que nous avons réalisées au fil du temps, avec l'ajout d'un suivi chronologique pour chaque soutenance. De plus, il s'est familiarisé avec le javascript, ce qui lui permis d'améliorer certaines animations, en rendant, par exemple, la barre de navigation dynamique ainsi que le suivi cronologique.

3.6 APIs

3.6.1 Compression et Décompression

Au départ, plusieurs fonctions ont été implémentées selon qu'il y avait un, des fichiers ou un dossier à compresser/décompresser. Mais ce qui manquait était le traitement de plusieurs dossiers. Au départ nous avons opté pour une solution assez brute. Le principe reposait sur le fait de créer un dossier qui regrouperait tous les dossiers à compresser, puis de compresser ce dossier-mère. Nous avions opté pour cette solution, car contrairement aux trois autres cas d'utilisations cités plus tôt, il n'existe pas de fonctions permettant de compresser plusieurs dossiers dans l'API SevenZipSharp, du moins explicitement.

En réalité, il existe dans cette API une fonction qui permet de résoudre ce problème, mais sous un autre angle de vue. C'est-à-dire que contrairement à avant, où on devait donner explicitement en paramètre le chemin du/des fichiers ou du dossier à compresser, cette fois-ci, il faut créer un dictionnaire qui a comme clé le chemin du fichier du point de vue de l'archive, et comme valeur le chemin du fichier sur le disque. Mais il a tout de même fallu créer de toute pièce un algorithme permettant de structurer tous les fichiers et sous fichiers au sein de l'archive.

De plus grâce à cette découverte, nous avions donc pensé à hériter les deux classes servant à représenter un fichier et un dossier par une classe mère. Grâce à ceci, nous avons pu réduire le nombre de fonctions nécessaire pour la compression et la décompression. Au lieu d'avoir une fonction pour compresser un/des fichiers, une fonction pour compresser un/des dossiers, en plus de ces mêmes fonctions pour l'extraction, nous avons désormais une fonction permettant de compresser une liste d'objets qui peuvent être un ensemble de fichiers et de dossiers, et une fonction permettant de faire l'extraction.

3.6.2 Fin du FTP

Il a tout d'abord été prévu d'implémenter entièrement une connexion FTP et FTPs, mais par manque de temps et une complexité très élevée, nous avons opté pour nous limiter à implémenter uniquement une connexion en FTP. Il a tout d'abord été question d'utiliser une API pour implémenter ce type de connexion. C'est ce qui a été fait, et l'API utilisée était FluentFTP. Au départ, il n'y avait aucun problème et les fonctionnalités marchaient bien, mais d'après plusieurs commentaires sur divers forums, cette API n'était pas adaptée pour une connexion FTPs.

Nous avons donc décidé d'utiliser des requêtes HTTP manuellement afin d'effectuer toute les opérations nécessaires sur un serveur FTP. Des classes ont notamment été créées afin d'avoir une représentation objet des dossiers et des fichiers distants pour pouvoir manipuler ces derniers avec aisance. Bien évidemment tout les paramètres ayant besoin d'intéragir avec des fichiers/-dossiers locaux ou distants utilisent des instances de classe et non des chaines de charactères représentant leurs chemins d'accès afin de pouvoir aussi les manipuler avec aisance.

3.6.3 OneDrive

L'implémentation de l'API Onedrive a été longue et compliquée. Nous avons tout d'abord cherché une API permettant de l'implémenter à notre solution facilement. Il en existe bien une, mais elle n'est compatible qu'avec le .Net Framework, alors que notre solution utilise la .Net Core 6.0. Ce qui obligeait cette API à utiliser la .Net Framework était uniquement avec la page de connexion à un compte microsoft afin d'accéder au service de stockage en ligne.

Toute les autres fonctionnalités étaient tout de même utilisables avec la framework que nous avons utilisée. Nous ne pouvions cependant pas nous permettre de changer de framework pour ceci car notre choix de permettre à Cubetools d'être crossplatform nécessite d'utiliser .Net Core.

Après de nombreuses recherches sur internet, nous nous sommes rendus compte qu'il fallait utiliser un tout nouveau genre d'API : une Rest API. Leur principe est que ses fonctionnalités ne sont accessibles qu'avec des requêtes HTTP.

Il était donc nécessaire de se référer à la documentation de Microsoft pour connaître l'utilisation et le format des URL pour accéder à toute les fonctionnalités nécessaires. Mais avant d'utiliser ces fonctionnalités, il faut tout d'abord récupérer un jeton d'accès à l'API avec Oauth 2.0.

Cette première étape a pris beaucoup de temps à cause du fait que la documentation de Microsoft à ce sujet est assez mal expliquée. Pour permettre à l'utilisateur de choisir son compte microsoft, nous avons opté par ouvrir un lien dans le navigateur par défaut, et ouvrir temporairement un port d'écoute. Ce dernier est primordial car lorsque l'utilisateur se connectera, il sera redirigé vers un lien local, qui contient le token d'accès, et qui sera capté par ce port d'écoute précédemment cité, puis fermé.

Après que cette première étape a pu enfin être franchie, l'implémentation des fonctionnalités à notre solution est allée relativement rapidement. Tout comme ce qui a été fait en FTP, il été nécessaire de créer des classes permettant de représenter les dossiers et fichiers sur Onedrive.

La spécificité avec ceci est qu'en récupérant les informations d'un objet Onedrive (fichier ou dossier), on récupère en réalité un json contenant plein d'informations. Nous avons donc utilisé ce que nous avons appris dans un TP pour la désérialisation d'un json en une instance de classe.

Pour l'upload de fichier, il faut envoyer une donnée spécifiant le type d'envoi à effectuer qui dépend de l'extension du fichier. Il était possible de lister tout les types de fichiers afin d'envoyer cette donnée nécessaire, mais ça aurait été un travail extrêmement fastidieux et extrêmement long. Fort heureusement, nous avons pu trouver une API qui permet de faire cette recherche : il s'agit de l'API MimeType.

3.6.4 GoogleDrive

Durant cette période, Max a réalisé la libraire Google Drive nécessaire à l'implémentation de cette fonctionnalité dans notre solution. Une partie de l'implémentation était aussi prévue pour la deuxième soutenance, mais le manque de temps et des complications sur d'autres points importants, nous ont obligé à repousser cette tâche à plus tard.

Tout d'abord, nous avons réalisé de nombreuses recherches pour nous familiariser avec l'API google drive, API que nous n'avions encore jamais utilisée auparavant. Il était nécessaire dans un premier temps de créer un projet sur la plateform google cloud, ainsi que demander de nombreuses autorisations pour accéder au drive de l'utilisateur par exemple. Tout cela est très bien expli-

qué dans la documentation de l'API, ce qui facilite grandement la tâche.

Une fois les fichiers identifiants téléchargés, nous avons pu nous attaquer aux différentes fonctions nécessaires pour notre projet, telles que la fonction upload, download, de recherche de fichier et bien d'autres. La documentation est plutôt bien détaillée pour comprendre comment manipuler l'API malgré le fait qu'elle ne soit pas en c. Nous avons pu aussi nous aider de nombreux sites qui présentaient l'API google drive.

De plus, elle nous donne accès à beaucoup de fonctionnalités différentes notamment en terme d'information disponible sur un fichier, on peut donc faire des sélections et recherches de fichier ou dossiers selon de nombreux critères (le type, la taille, la date de création, si une lettre, un mot est présent dans un fichier, etc...). Enfin, chaque dossier détient un identifiant unique qui lui est propre ce qui permet d'accéder et de manipuler facilement un fichier.

Nous avons tout d'abord créé les fonctions basiques pour créer un fichier ou un dossier, ainsi que les fonctions de recherche de fichiers ou d'informations sur un fichier, ainsi que la modification de celui-ci (renommer par exemple). Ces fonctions sont assez basiques et ne nous ont pas posé trop de problèmes.

Nous avons néanmoins rencontré quelques problèmes lors de la confection des fonctions download et upload, qui sont un peu plus difficiles.

3.7 Fonctionnalités supplémentaires

3.7.1 SnapDrop et Smash

Pour cette deuxième période et pour la fin du projet, nous avons décidé d'intégrer ces fonctionnalités (qui sont des bonus) et de

faire en sorte que lorsque l'on clique sur le bouton, votre navigateur par défaut est ouvert sur la page de la fonctionnalité choisie !

Cela est plus simple pour nous, mais surtout pour l'utilisateur qui n'aura pas à se soucier avec notre interface qui serait différente de celle du site (avec des champs qui sont liés au site via des requêtes HTTP en POST).

3.7.2 Fichier de configuration en JSON

Un système de configuration chargé dans le code C# a été proposé par Matéo.

Ce système de configuration n'est rien d'autre qu'une classe qu'on vient désérializer à partir d'un fichier JSON dans une variable à l'aide de DataContract et de DataMember

Lors de cette période, nous avons mis en place un fichier de configuration simple capable de stocker quelques informations comme le chemin de notre logiciel ou bien le chemin du JSON chargé dans notre configuration

Nous avons ainsi créé à la fois le désérializer mais également le sérializer qui permet de faire l'opération inverse. On vient transformer la classe en un fichier JSON. Pour ne pas générer des milliers de fichier JSON à chaque fois que l'on crée ou sauvegarde la configuration, on récupère le chemin du JSON chargé puis on vient le détruire et réecrire son contenu.

4 Vers un code modulaire et Performant : 3ème période

De nombreux problèmes sont apparus après la seconde soutenance de notre projet, notre implémentation devenait de plus en plus un frein.

L'implémentation nous limitait beaucoup, il y avait un grand nombre de popup et le code n'était pas vraiment modulaire.

4.1 Workers et Threads : La solution pour refresh

Afin de rendre compte du'une modification en direct, plusieurs se sont offerts à nous :

— Utilisation de notification : utilisé dans le modèle MVVM, il permet de notifier le changement d'une variable et d'actualiser en direct l'interface utilisateur. Cette solution n'a pas été retenue et n'a pas été utilisé car notre modèle ne reposait pas sur le modèle MVVM mais bien sur le modèle MVC

— Utilisation des events : Nous avons utilisé de nombreuses fois les events dans le code-behind. Ils sont pratiques et optimisés car nativement pris en charge par Avalonia.

— Utilisation de Workers : les workers sont des threads qui tournent en arrière-plan et qui modifie en temps réel le UI ou bien des variables.

Pour la dernière période de ce projet, il fallait par exemple mettre à jour en direct les drives actuellement connectés à l'OS de l'utilisateur. Pour cela, nous avons utilisé un worker qui vient rafraishir régulièrement les drives si leur nombre change par exemple

Le même principe de fonctionnement a été utilisé pour mettre à jour les liens favoris de l'utilisateur et pour changer en direct le thème sombre ou clair de l'interface.

4.2 Vers un code modulaire et abstrait

Un nouveau système de Client a été développé par Mehdi et Matéo lors de cette dernière période

Le principe de cette nouvelle implémentation repose sur un principe simple. Au lieu de créer une vue différente par type de client, on vient créer une vue générale qui représente un client et on vient utiliser un client abstrait permettant d'implémenter les fonctionnalités du GoogleDrive, du FTP, du OneDrive et du Local dans un seul et même client.

Pourquoi ce changement ? Il était necessaire pour nous de réaliser un code le plus modulaire possible étant donné le nombre de popup qui nécessite des références vers ce client abstrait. Si ce dernier était différent, il aurait fallu multiplier le nombre de popup à créer par 4 sans parler des problèmes de converssion.

4.2.1 Client abstrait

Le problème qu'on avait avec notre première implémentation des différents clients, que ça soit en local ou en distant, était qu'on avait un bout de code pour chaque client, et pour chaque fenêtre. Le problème avec ceci était la répétitivité de code et surtout la répétitivité de fonctionnalités.

Il existe de nombreux points commun entre tous ces clients : ils possèdent une barre de navigation, quasiment les mêmes boutons d'action, et un panneau qui affiche une liste d'élements qui sont un ensemble de fichiers et de dossier.

Nous avons donc procédé à une refonte du code pour ces fonc-

tionnalités. On a tout d'abord créé une classe abstraite qui est parent de tous les types de clients : Local, FTP, OneDrive et GoogleDrive. La fenêtre principale, ou la fenêtre d'accès à distance n'a qu'à récupérer le Client à afficher.

Le cas des boutons d'actions est assez spécial, car ils partagent tous des mêmes fonctionnalités, mais selon le Client dans lequel ils sont affichés, ils ne sont pas tous les mêmes. Il a donc fallu faire en sorte d'écrire un seul code pour les boutons, mais qu'on puisse avoir le choix desquels on veut ajouter à un Client.

Pour ce faire, une classe abstraite qui représente un bouton d'action a été créée. Ensuite, chaque bouton qui représente une action spécifique a été créée, mais chacune hérite de cette classe abstraite créée précédemment. Finalement une liste de boutons d'action est donnée au démarrage de l'affichage du client afin de les afficher dans la barre d'action.

Il fallait aussi créer un lien de parenté entre tout les éléments qui représentent des fichiers/dossier en local, FTP, OneDrive et GoogleDrive. Une classe abstraite Pointer a été créée, et toutes les classes, que ce soit en local ou en distant, héritent désormais de cette classe abstraite.

Avec toute ces modifications faites, toutes les conditions nécessaires afin de mutualiser tous les clients.

4.2.2 Boutons abstrait(e)s

Plus tard, Mehdi a implementé un système de bouton modulaire

Le principe est simple : lorsque Mehdi et Matéo ont mis en place le Client dans le UI, il n'avait pas prévu le fait que certains boutons puissent être présents, nous avions littéralement une copie parfaite du UI dans la fenêtre distante

Mehdi s'est donc occupé de rajouter des boutons lors de la défi-

nition de la Window (local ou distante) en fonction du type de client utilisé.

Par exemple, en FTP, les boutons copie, paste et cut sont inexistants alors qu'ils sont présents dans OneDrive.

4.3 UI Final

Pour le UI, de nombreux changements on été effectués pour avoir une application la plus modulaire possible. En réalité, l'équipe avait une implémentation qui devait avoir des popups dédiées pour chaque action, c'est à dire chaque popup gère uniquement la partie locale, avec une dixaine de popup d'erreur.

Comme expliqué précédemment, notre application avait besoin de passer sur le modèle en MVC mais malgré celà, nous avions besoin de passer sur un code modulaire pour rendre l'application bien plus légère. Nous avons donc réfléchi à faire l'implémentation en full-modulaire. C'est à dire avoir une vue abstraite (comme les classes abstraites en C# décrites précédemment). (voir CubeTools Final)

Pour ce faire, nous avons dû réadapter toute l'implémentation du UI. c'est à dire supprimer toutes les popups/fenetres du Ui pour ne faire que des classes abstraites.

Nous avons commencé par faire une seule fenêtre et ne mettre que des références au code C#. Par exemple, pour l'Actionview (la partie avec les boutons d'action sur les fichiers et dossiers), nous avons juste une barre vide à l'affichage. Et dans le code C#, on définit les boutons à instancier. Le problème est que cette génération doit être minutieusement définie par des variables dans le code AXAML et gérer déjà l'affichage de boutons statiques générés, pour ensuite rafficher les bonnes popups et plus personnalisées par conséquent. Pourquoi ? Car nous avons accès à toutes les variables dans le code C#.

Par exemple, on peut créer un fichier usercontrol en AXAML qui permet d'être généré et intégré dans n'importe quel conteneur XAML (par exemple : les stackpanels, les wrapspanel, les grids etc)

Kévin s'est donc occupé de tout refaire pour être sûr que le générateur place bien les boutons dans le UI. Il a commencé par revoir toute la version non dynamique qui présentait des bugs dans certaines fenêtres ou dans des popups.

Il en a profité avec Matéo pour refaire l'arborescence des icônes pour qu'elle soit sous forme de loaders et donc de iconspacks. Nous avons fait cela car le JSON expliqué précédemment permet de savoir ce que nous sauvegardons ou non, y compris le pack d'icone. Avec le JSON, il est donc possible de générer un pack d'icone à l'ouverture de l'application.

Avec cette nouvelle arborescence, il est possible de créer ses propres pack d'icones. Par exemple, pour créer un pack d'icones il suffit de copier coller le dossier Default et de faire ou importer les icones souhaitées. Par la suite il sera possible de modifier les couleurs de l'application, c'est à dire les couleurs de sélection, de désélection, de barres de chargement etc.

Pour en revenir sur les fenêtres abstraites, une fois réalisées, il fallait de nouveau tout relier au code C#. Mehdi et Matéo ont aidé Kévin à tout relier dans le code.

Comme expliqué précédemment, nous avons donc une immplémentation entièrement modulaire et capable de s'adapter à de prochaines fonctionnalités/fenetres/classes.

Pour finir, Kévin devait gérer les boutons/fonctions générées avec le code C#. Mehdi et Matéo ont lié la partie backend avec les nouveaux boutons abstraits.

L'équipe s'est donc mobilisée pour faire énormément de de-

bug tant au niveau de l'affichage que dans le code directement. Avant, le debug était simple car les fonctions en C# et le code AXAML n'étaient pas liées. Nous avons donc dû faire de grosses sessions de debug et radapter le code petit à petit pour ne plus faire planter l'application.

Après avoir réimplémenté les fonctions, Kévin devait se charger de refaire tous les icônes du menu en Dark pour avoir un thème sombre et clair lorsque l'application passe d'un thème à un autre. Il faut donc refaire tous les icpines de l'affichage de l'application (pas les extensions de fichiers).

Pour cela, encore une fois, PhotoEditor a été utilisé mais cette fois-ci, les icônes qui ont déjà été faites ont dû être refaites. Pour ce faire, Kévin a fait en sorte que les boutons se voient sur le noir tout en restant dans les teintes jaunes, tâche complexe si l'équipe veut en plus récupérer les mêmes icônes !

Tout d'abord il a fallu réimporter le dossier et faire les icones une par une. De longues sessions de recherches ont été nécessaires pour trouver les bonnes couleurs pour finalement arriver à un résultat très convaincant ! Dans ce cas, Nous avons appliqué d'abord filtre d'invesion de couleurs pour avoir le bouton noir qui devient blanc, le jaune devient bleu. Mais en ajoutant par dessus un ajustement de couleurs Hue réglé au minimum, nous retrouvont le jaune tout en gardant le blanc du bouton désiré (car celui doit bien apparaitre dans le UI sur fond noir).

4.3.1 Configuration et JSON

Cette partie est d'autant plus intéressante car elle concerne la personnalisation de l'interface et des préférences de l'utilsateur. Ces fonctionnalités sont disponibles grâce à notre nouvelle implémentation.

Pour commencer, la fenêtre "paramètres" inclue plein de fonc-

tionnalités de personnalisation mais également l'option désinstaller (que vous ne ferez jamais). Les paramètres incluent la personnalisation du thème, les raccourcis, les favorsi et serveurs FTP.

Pour commencer avec la partie thème, il suffit de choisir dans le menu déroulant le thème que l'on veut choisir. Ils peuvent être ajoutés dans le dossier Assets. Pour éviter tout problème de plantage, il est conseillé de copier les dossiers Default comme décris précédemment. Pour fonctionner, les thèmes sont chargés à partir d'un watcher qui regadre les dossiers présents dans Assets. Lorsque le thème est saved, le JSON est réécrit pour mettre le nouveau thème appliqué. Il déserialize le fichier, l'utilsiateur chosiit le thème et lorsqu'il est save, il est Serialized, c'est à dire réécrit.

Dans le menu de l'application ou dans la Linkbar, sont présents les favoris. Ceux-ci sont extrèmement pratiques pour avoir un accès direct aux dossiers/fichiers importants ou pour les retrouver facilement lorsque le chemin d'accès est fastidieux. Ils sont modifiables via les paramètres avec la rubrique associée. Elle donne une liste des favoris qui peuvent être modifiés en nom et en localisation. Pour les ajouter ou supprimer directmeent depuis le UI, il suffit de cliquer sur l'étoile, elle se teindra en jaune.

Cela est très pratique car l'utilisateur peut les modifier en temps réel et les sauvegader, les gérer.

Pour modifier les raccourcis, il faut cliquer sur le raccourci en question puis cliquer sur les boutons pour activer le raccourci. Puis cliquer sur save pour suavegarder. Nous avons fait le meme principe que pour les favoris et les thème.

Enfin, les serveurs FTP peuvent être modifiés dans cette rubrique en cliquant sur le menu déroulant. Encore une fois, grace

a notre implementation et au fichier JSON, nous pouvons faire un système de sauvegarde de raccourcis.

Par ailleurs, la configuration peut etre exportée d'un cubetools à un autre simplement en copiant le fichier Config.json ou bien en générant un fichier dont ayant un pattern valide et dont le nom commence par 'Config'.

4.4 Website

Max a réalisé durant cette période, les parties qui nécessitaient une avancée conséquentee de notre projet dans sa globalité. C'est à dire, la présentation de notre projet ainsi que des services proposés. En effet, l'interface graphique a été réalisée au fil du temps avec beaucoup de changemente par rapport au premier jet. De plus nous n'étions pas certains des différentes fonctionnalités qu'il y allait avoir dans notre solution jusqu'à assez tardivement. Max a donc réalisé à la toute fin le caroussel d'images ainsi que la grid présentant toutes les fonctionnalités.

Max a aussi rajouté plusieurs animations pour rendre le site web plus fluide et attractif. Enfin, il a intégré les boutons permettant de télécharger la version adéquate pour son système d'exploitation (Windows, Mac, Linux), ainsi que le manuel d'utilisation et les ressources utilisées.

Max n'a pas rencontré beaucoup de problèmes durant la création du site, mais une réflexion assez longue a été nécessaire afin de présenter efficacement le projet tout en gardant le design initial de la page web.

4.5 APIs

4.5.1 OneDrive terminé

La majorité de la librairie one drive a déjà été implémentée. Il ne restait plus qu'à implémenter certaines fonctionnalités oubliées, comme par exemple le Rename pour renommer un fichier ou dossier, ou modifier des fonctions déjà implémentées mais mal adaptées au projet, par exemple la fonction GetItem qui permet de récupérer une instance d'un Item (Fichier ou Dossier) à partir d'un path donné en paramètre.

Il était nécessaire de corriger de nombreux bugs à cause de la liaison de la librairie avec l'interface graphique. Par exemple, le chemin d'accès racine sur Onedrive se note "/drive/root :". Les sous dossiers et sous fichiers s'ajoutent après ce dernier. Mais afficher cette racine dans le champs de texte qui affiche le chemin d'accès n'est pas très user-friendly. Il a donc fallu faire en sorte d'afficher dans ce champs de texte uniquement ce qui suit cette racine, tout en backend, prendre en compte le chemin d'accès au complet.

Il a aussi fallu régler des bugs liés à une désynchronisation de l'affichage des fichiers/dossiers. Par exemple si un fichier a été supprimé dans le dossier courant, mais depuis un navigateur web par exemple, la liste des éléments ne sera pas actualisée automatiquement. Donc si l'utilisateur décide de supprimer ce fichier qui a déjà été supprimé, il ne faut pas que l'application crashe. A la place, l'affichage du dossier courant est actualisé.

Finalement, avec l'ajout de parenté pour la représentation objet les fichiers/dossiers onedrive afin d'avoir un parent commun pour tout type d'objet, que ça soit en local, FTP, one drive ou google drive, il a fallu réassigner les variables sur les nouvelles variables appartenant à cette classe parent.

4.5.2 GoogleDrive terminé

Une grande partie de la librairie google drive étant déjà faite Max n'avait plus que l'implémentation à réaliser, nous avons néanmoins dû réadapter certaines fonctions pour satisfaire la création de classe abstraite en cohérence avec la structure du FTP et de One Drive qui était en arborescence, tandis que notre implémentation reposait sur le principe d'identifiant unique pour chaque fichier, nous avons donc créé une fonction qui permet de récuperer cet identifiant à partir du chemin d'accès de ce fichier, ainsi que la fonction inverse (chemin d'accès à partir de l'identifiant).

Ensuite, nous avons testé les fonctions qui ne l'étaient pas encore, nous avons rencontré plusieurs problèmes : le dossier parent était plus difficile d'accès que nous le pensions, nous avons eu besoin de modifier les fonctions qui nécessitaient la lecture de l'identifiant du dossier parent d'un fichier. De plus la modification d'un fichier nécessite certains droits que nous ne pouvions pas avoir avec la façon dont nous avions réalisé nos fonctions, nous avons dû trouver une solution pour les obtenir, solution que nous avons trouvée après plusieurs recherches sur internet.

L'implémentation de l'API google drive s'est plutôt bien déroulée malgré le non fonctionnement de certaines fonctions dans un premier temps pour diverses raisons, on a remarqué les différents problèmes assez rapidement et les solutions n'étaient pas très difficiles à trouver. Il s'agissait principalement de faire remonter les erreurs que l'on pouvait rencontrer lors de l'utilisation de notre explorateur.

Cependant, nous avons rencontré un problème de taille : la connection entre l'API et notre solution était extrêmement lente, il fallait plusieurs secondes pour créer un nouveau fichier ou bien seulement pour utiliser l'outil renommer (pouvant aller à

plusieurs minutes pour l'affichage de dossier très volumineux). Nous avons tout d'abord essayé d'optimiser notre code au maximum, en essayant de minimiser le nombre de requêtes à l'API, malgré cela on remarquait très peu de changements en terme de performance. Nous nous sommes donc penchés sur les solutions que nous pouvions trouver sur internet, nous avons tout de suite remarqué que nous n'étions pas les seuls à avoir des problèmes de performance. Les réponses étaient peu convaincantes, et que le temps étonnamment long était dû au nombre de personnes utilisant le même serveur et notre connection internet. Enfin, ils nous rappellent pour la plupart que "cette API est gratuite et que l'on n'obtient que ce pour quoi on a payé". Néanmoins, nous avons fini par trouver, un moyen d'afficher efficacement la liste des fichiers et dossiers d'un dossier parent, ce qui rend l'expérience utilisateur bien plus intéressante, malgré un temps un peu inférieur à nos attentes initiales pour certaines tâches, l'implémentation de google drive est utilisable dans l'ensemble.

Il ne restait ensuite qu'à implémenter les différentes exceptions auquel l'utilisateur peut être confronté lors de son utilisation. Cela n'a pas pris beaucoup de temps grâce aux différentes Pop up déjà créées pour la partie locale de notre application.

Derniers petits problèmes, pour rendre le projet google cloud publique, il fallait faire vérifier notre application, étape compréhensible, l'application va gérer des données privés des différents utilisateurs de notre solution. Seulement, pour faire vérifier une application, il était nécessaire de fournir une vidéo youtube, ainsi qu'un rapport écrit expliquant notre projet, comment on allait gérer les données des utilisateurs et bien d'autre. Ce qui demandait beaucoup de temps, pas assez pour gérer l'arrivé de la soutenance. Le projet reste donc en phase de test pour le moment, mais peut-être néanmoins utilisé.

5 Avis

5.1 Max

J'ai beaucoup aimé réaliser ce projet, avec mes collègues. Cela m'a permis de découvrir une partie essentielle de notre métier futur, les projets communs. C'est la première fois que je travaillais en groupe sur un si gros projet, il fallait donc beaucoup de rigueur, de coordination pour s'entendre sur les tâches à faire ainsi que la direction vers laquelle se tournait le projet. Dans l'ensemble, nous n'avons eu aucun désaccord, l'avancée du projet n'a donc pas été dérangée par une mauvaise ambiance de travail. Nous nous sommes très bien organisé, ce qui rendait le projet d'autant plus intéressant.

De plus, pour ma part, j'ai été confronté à de nombreuses premières. C'était la première fois que je touchais au développement web et j'ai beaucoup apprécié, apprendre et m'améliorer sur le sujet. J'y ai passé beaucoup de temps, ce qui m'a permis d'apprendre beaucoup de choses, au fil de l'avancée du site j'ai pu constater une réelle progression, une amélioration du site web.

L'utilisation d'Avalonia était aussi une première, au même titre que WPF, la plateforme de développement qui a beaucoup inspiré les développeurs d'Avalonia, il y avait beaucoup de nouvelles notions à prendre en main, et le peu de documentation présente sur internet n'ont pas beaucoup aidé. Néanmoins, grâce à notre travail collectif, l'application s'est développée petit à petit et la progression est nette.

Enfin, l'implémentation de Google Drive a été très intéressante et bénéfique pour moi. J'ai découvert la plateforme de cloud computing de google, qui m'a permis entre autre de créer un projet pour pouvoir utiliser l'API google drive. J'ai donc passé beaucoup de temps à me former sur les différentes fonctionna-

lités que proposait leur API et leur espace développeur dans sa globalité.

Nous n'avons pas pris le projet le plus facile à réaliser, et les problèmes rencontrés ont été nombreux, mais j'ai pris beaucoup de plaisir à chercher des solutions, améliorer notre projet et bien d'autres choses. Nous avons appris beaucoup de choses qui vont forcément nous servir dans le futur, que ce soit en cherchant sur internet ou bien avec l'aide de mes collègues de travail qui m'ont beaucoup apporté tout au long du projet grâce à leurs connaissances plus approfondies dans certains sujets que les miennes.

J'ai bien compris au fil de l'avancée que le travail d'équipe est la clé d'un projet réussi !

5.2 Mehdi

J'ai adoré travailler en groupe sur ce projet. Au départ on avait le choix entre soit faire un jeu vidéo ou un logiciel. Personnellement, les deux types de projets pouvaient me convenir.

Dans le premier cas, cela aurait été un peu plus amusant, tout en découvrant un environnement de travail autour d'un jeu vidéo. Ce qui est spécial avec ce dernier point c'est qu'il y a un brassage de créativité et de technicité entre la réalisation des modèles 2D/3D, l'écriture de l'histoire, l'élaboration des intelligences artificielles... En revanche dans le deuxième cas, l'environnement de travail allait être assez différent.

Je programmais à titre personnel depuis plusieurs années, mais je n'ai rencontré absolument personne qui connaissait ce domaine. Je n'avais donc absolument aucune idée de ce que ça pourrait être de programmer un même projet en groupe. Désormais, grâce à ce projet, je sais ce que ça fait, et j'ai adoré. Les tâches peuvent être réparties chez plusieurs personnes et les décisions pour implémenter les fonctionnalités sont débattues.

J'utilisais déjà git auparavant, mais puisque j'étais seul sur une repository, l'intérêt était peu présent. Mais avec ce projet, j'ai pu pousser la quantité de fonctionnalités utilisées assez loin.

De plus, j'ai découvert un nouveau concept à appliquer à la programmation. De manière générale, j'ai toujours programmé sur de l'orienté objet. En implémentant un client OneDrive à CubeTools, j'ai découvert le fonctionnement d'une API. Il existe déjà une API préfaite, mais qui n'est pas compatible en cross-platform. J'ai dû donc y accéder avec des requêtes HTTP. Pour ce faire, ça m'a forcé à lire de manière complète une documentation, chose que je ne faisais que très peu ou pas auparavant.

En somme, ce projet est, de mon point de vue, un premier pas vers le monde professionnel. Il nous a permis d'apprendre à travailler, se coordonner et communiquer en groupe de travail.

5.3 Kévin

Pour ma part CubeTools était un projet très enrichissant. Depuis le tout début, avec mes collègues, notre idée était claire : faire une application plutôt qu'un jeu vidéo.

Nous avons donc tout de suite pensé à un explorateur de fichiers ! Mais pas n'importe lequel, un explorateur fait pour faciliter le partage de fichiers et ce de manière cross-platform !

Pour parler du projet ou plutôt le groupe, il y avait une super entente du début du projet jusqu'à la fin. Notre organisation a été très efficace ce qui rendait nos sessions de travail très productives et motivantes !

Vraiment, j'ai pris énormément de plaisir à travailler sur ce projet. D'une part parce que créer son propre explorateur de fichier est très enrichissant : il faut penser à tout ! L'optimisation, les algorithmes de recherche, les erreurs, le design, la mise en page, les APIs, tout y passe !

Pour ma part, j'avais vraiment l'impression de travailler sur un projet qui va servir tant personnellement que professionnellement. Notamment parce que notre application est très fonctionnelle, mais aussi parce que j'ai énormément appris tant sur la partie code que créative.

Sur la partie code, ce n'était pas une tâche facile du tout, j'ai dû énormément apprendre sur le tas et me documenter, chercher des astuces, des méthodes pour coder ce que je voulais faire. Des heures de recherches et de travaux pour arriver à construire ne serait-ce qu'une fenêtre ou popup et le lier au code C#. Mais malgré tous nos efforts, nous sommes très grandement satisfaits de notre application car elle correspond exactement à ce que nous recherchions !

Sur la partie créative, il était très difficile pour moi, malgré l'ex-

périence que j'ai acquérir ces dernières années en design, il était relativement difficile de trouver un style et des couleurs qui correspondaient à l'image de CubeTools, refaire les icones, les fenêtres jusqu'à ce que l'application soit enfin satisfaisante visuellement ! Des heures de recherche sur les designs à la mode, ce qui plairait à l'utilisateur et tout ce qui est possible de faire pour un explorateur de fichiers et avec la framework Avalonia.

Ce projet était donc sûrement le projet le plus enrichissant de ma vie. J'ai pris énormément de plaisir à partager ce travail avec mes collègues. Bref, le meilleur projet que j'ai pu faire de ma vie.

5.4 Matéo

J'ai réellement apprécié de travailler avec notre équipe sur le développement de notre logiciel CubeTools. J'ai vraiment eu l'impression d'avoir proposé un produit professionnel.

Travailler sur ce projet m'a permis de m'enrichir sur plusieurs points :

D'une part sur le travail d'équipe et sur l'utilisation de git. On n'y pense pas souvent mais il y a pleins de fonctionnalités cachées autre que push, commit, add et pull. En travaillant sur ce projet, j'ai pu découvrir comment travailler efficacement sur un projet à plusieurs notamment grâce aux réunions que nos avons pu mettre en place mais également lorsqu'il s'agissait de séparer les tâches.

J'ai également développé des compétences sur ma façon de programmer avec notamment les classes abstraites et l'implémentation modulaire que moi-même et Mehdi avons proposées afin de rendre le code plus solide. Ma façon de programmer a aussi évolué puisque j'ai appris à écrire et à utiliser une libraries que j'ai moi-même créées (la librarie pour le client local).

Je n'oublie pas toutes les choses que j'ai pu apprendre lorsque j'ai implémenté l'interface utilisateur. J'ai eu l'occasion de refaire une grande partie du UI avec différents modlèles (MVVM puis MVC). J'ai pu en partie découvrir comment programmer une interface et comment la rendre fonctionnelle.

En conclusion, je retiens les nombreuses heures passées sur ce projet, les temps d'échange autour d'un bug, la rage qu'a pu provoquer les bindings au début du projet.

Merci à l'équipe !

6 Annexes et Sources

6.1 Logiciels et Applications

Background Eraser (Android)

Photo Editor (Android)

Rider

Visual Studio

Visual Studio Code

6.2 Images

UI bleu nouveaux icones nouveau UI popup renommer icones compresses

6.2.1 Deux premières versions du UI

Flaticon

Google Images

6.2.2 Version Finale

Pour rappel, toutes les images et icônes dans la version finale du projet ont été entièrement réalisées par l'équipe KM3 !

6.3 Documentations et ressources

6.3.1 YouTube

Chaines :

JDsCodeLab

C Design Pro

Payload

Implémentations :

Code-Behind to MVVM

6.3.2 Docs

AvaloniaUI

MSDN - Microsoft C# documentation

Documentation Onedrive Rest API

6.3.3 Forums

MSDN - Forum

GeeksForGeeks StackOverflow

AvaloniaUI - GitHub

6.4 Images du rapport

Figure 1 – Logo KM3

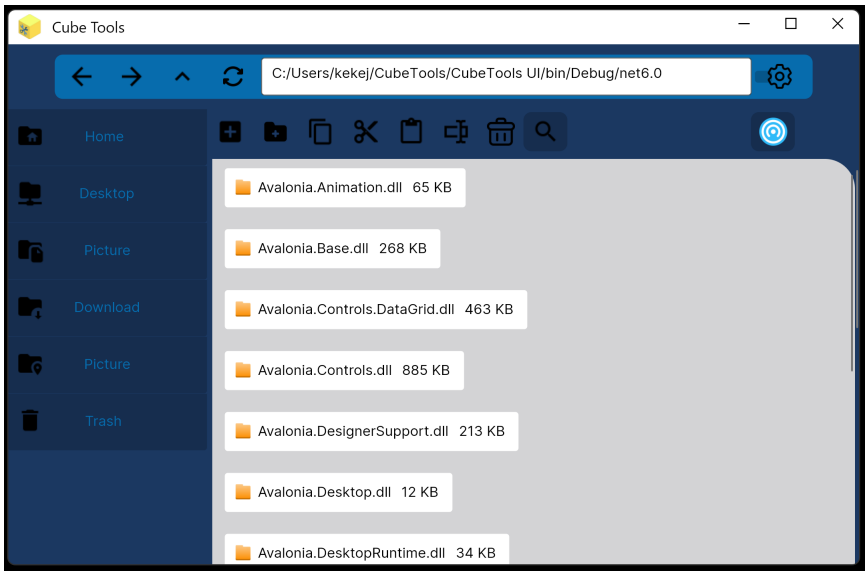


Figure 2 – 1er UI bleu

	1er Soutenance	2e Soutenance	3e Soutenance
Site Web	**50%**	**80%**	**100%**
Logiciel			**90-100%**
UI	40%	80%	100%
Basiques	80%	100%	-
Avancés	0%	30%	80-100%
Transfert			**90-100%**
FTP	20%	100%	-
SFTP	0%	100%	-
Autres méthodes	0%	30%	70-100%
Cloud			**90-100%**
API OneDrive	30%	60-100%	100%
API GoogleDrive	-	30%	80-100%
Bonus	**Snapdrop - Smash**	**NearbySend**	**Android et IOS**

Figure 3 – Tableau théorique de l'avancement

Figure 4 – Icône WPF

Figure 5 – Icône GTK

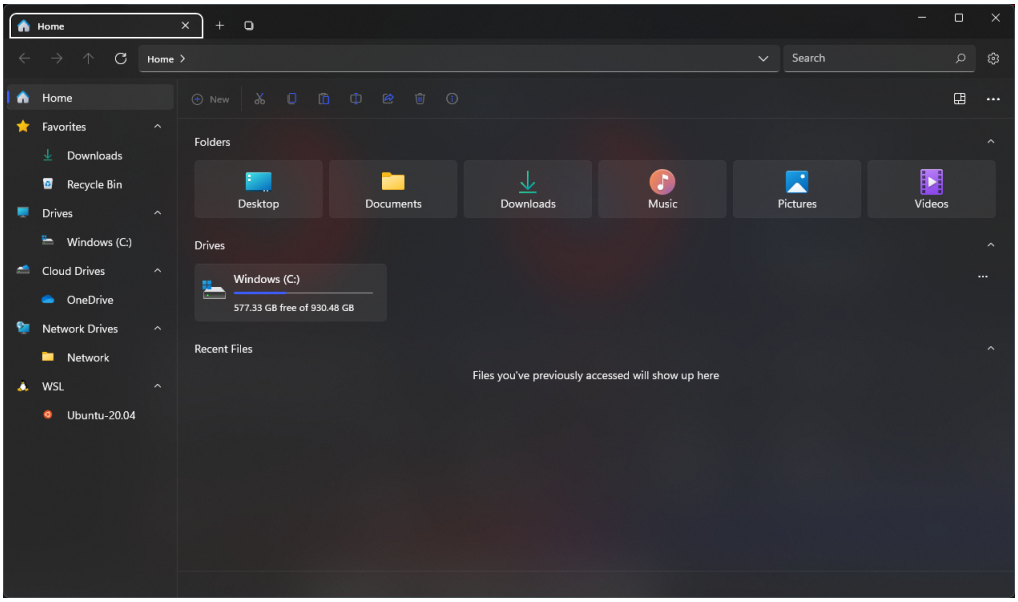


Figure 6 – Interface de Files

Figure 7 – Icône CXScreen

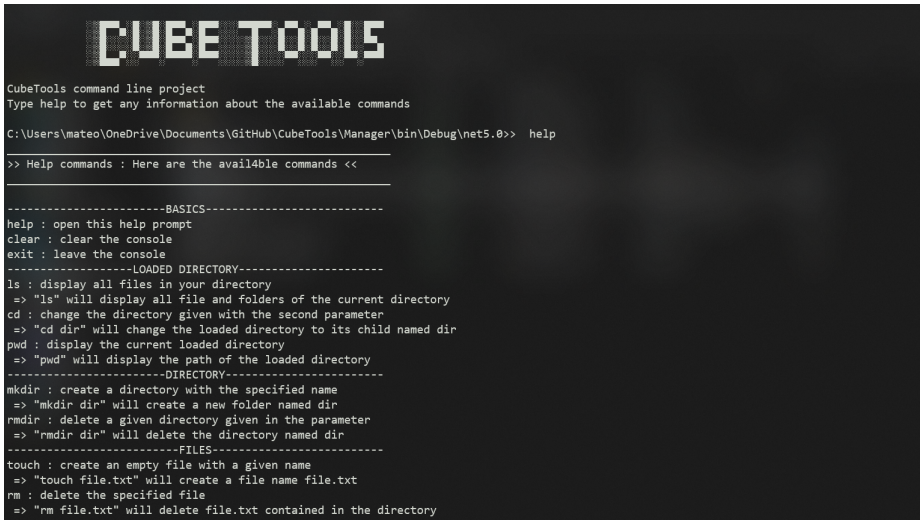


Figure 8 – Screenshot CLI

```
refreshing directory : C:\Users\mateo\OneDrive\Documents\GitHub\CubeTools\Manager\bin\Debug\net5.0
C:\Users\mateo\OneDrive\Documents\GitHub\CubeTools\Manager\bin\Debug\net5.0>>  mv test tes1
########################################
###         Path does not exist       ###
 # Low : PathNotFoundException
 # error : the given path tes1 does not exist
 # error at : GenerateNameForModification
########################################
```

Figure 9 – CLI Errors

Figure 10 – Error Popup

Versions	Coûts Mémoire	Coûts Calculs
UI 1ère soutenance	400-600Mo RAM	Normale
1ère Restructuration	400-600Mo RAM	Lourde
2ème Restructuration	500-800Mo RAM	Très lourde
Dernière implémentation	150-250Mo RAM	Légère

Figure 11 – Avancement de l'optimisation

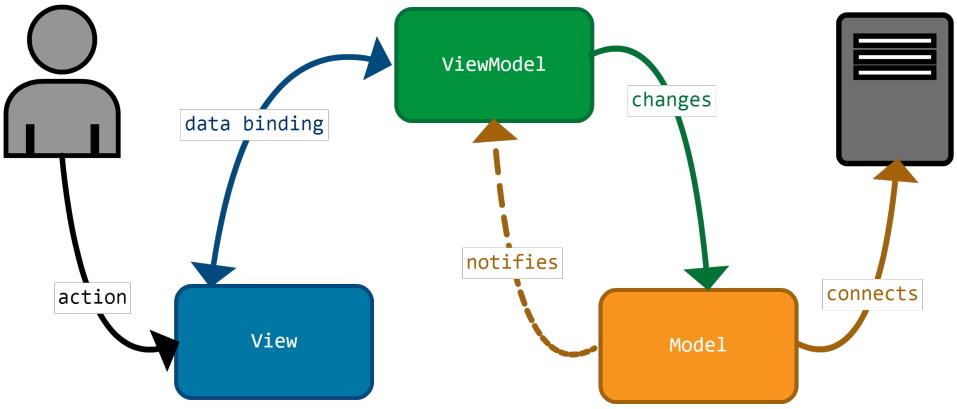


Figure 12 – Modèle MVVM

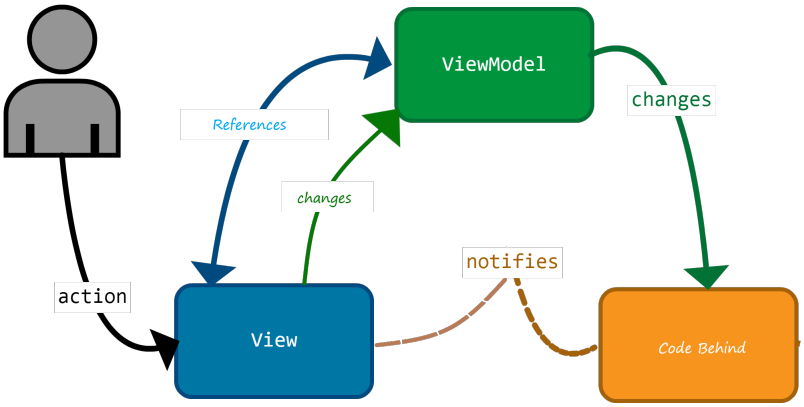


Figure 13 – Modèle MVC

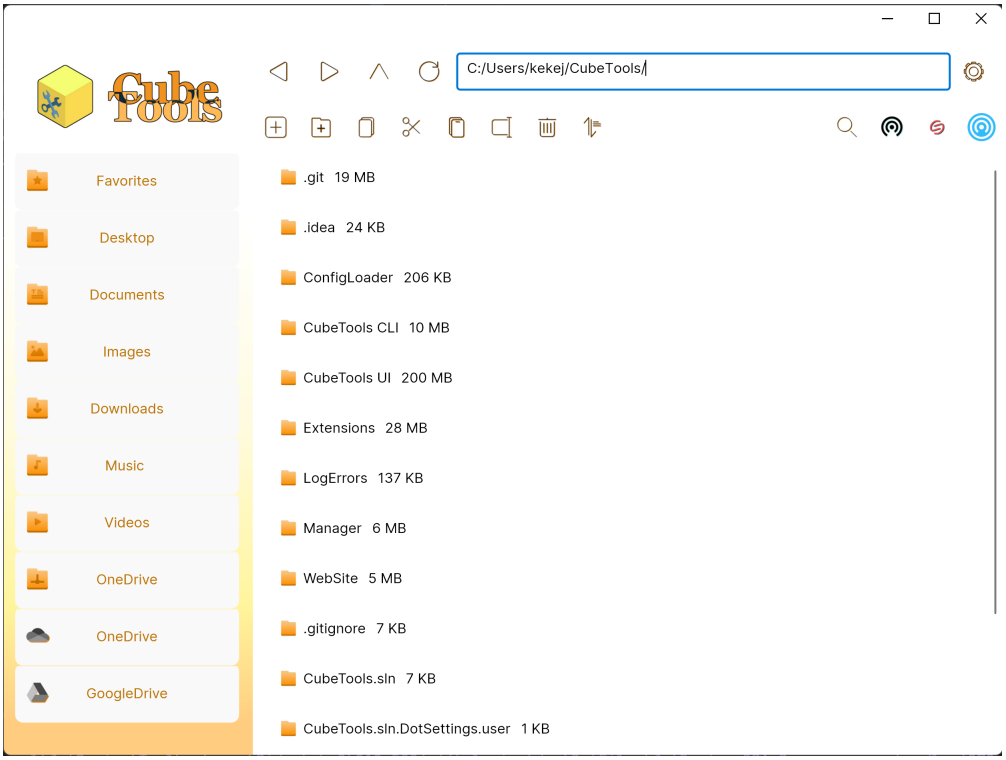


Figure 14 – CubeTools Soutenance 2

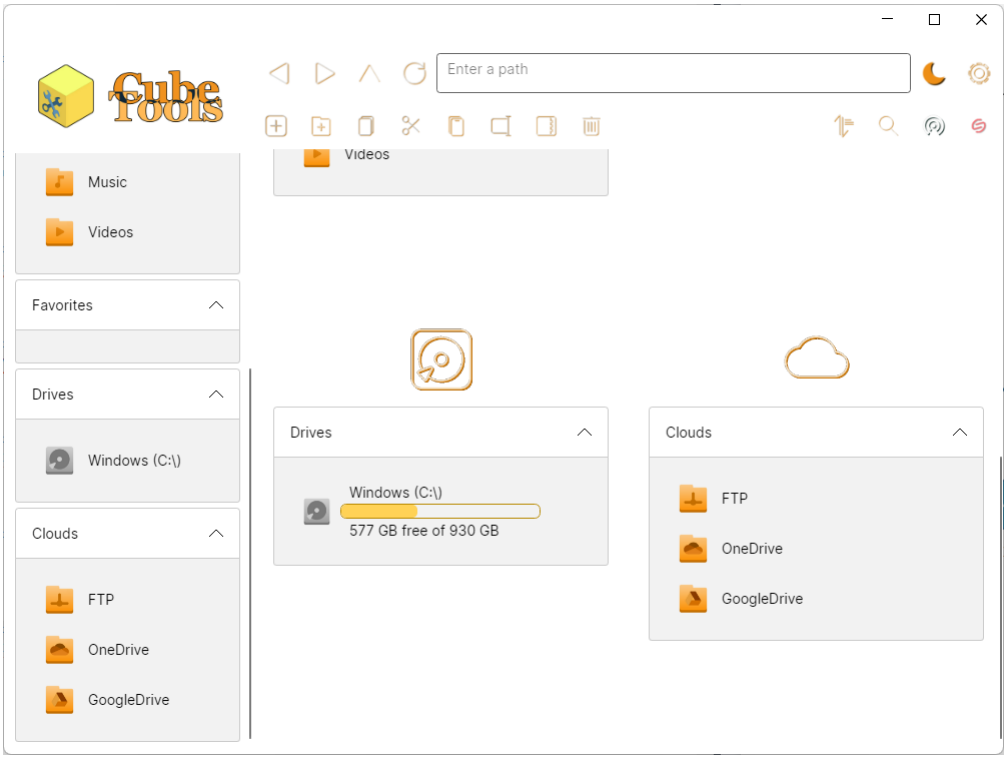


Figure 15 – CubeTools Final