Rapport de TPI - MailStorage

Maziero Marco – CIN4A

Lausanne – ETML - 2017

Durée : 5.5 semaines

Chef de projet : Melly Jonathan - [jonathan.melly@vd.educanet2.ch](mailto:jonathan.melly@vd.educanet2.ch)

Expert N°1 : Montemayor Ernesto - ernesto@bati-technologie.ch

Expert N°2 : Moren Auxilio - auxilio.moren@orif.ch

Table des matières

[1 Spécifications 8](#_Toc484598753)

[1.1 Titre 8](#_Toc484598754)

[1.2 Description 8](#_Toc484598755)

[1.3 Matériel et logiciels à disposition 8](#_Toc484598756)

[1.4 Prérequis 8](#_Toc484598757)

[2 Planification Initiale 10](#_Toc484598758)

[2.1 Liste des tâches 10](#_Toc484598759)

[2.2 Diagramme de Gantt 10](#_Toc484598760)

[3 Analyse 12](#_Toc484598761)

[3.1 Opportunités 12](#_Toc484598762)

[3.1.1 Utilisation du protocole IMAP 12](#_Toc484598763)

[Solution 12](#_Toc484598764)

[3.1.2 Stockage de fichiers sur boite mail 13](#_Toc484598765)

[Solution 13](#_Toc484598766)

[3.1.3 Approfondissement des connaissances 14](#_Toc484598767)

[3.2 Document d’analyse et conception 15](#_Toc484598768)

[3.2.1 Architecture du projet 15](#_Toc484598769)

[Classe LoginWindow 16](#_Toc484598770)

[Classe DbManager 16](#_Toc484598771)

[Classe MailStorage 16](#_Toc484598772)

[Classe statique MailManager 16](#_Toc484598773)

[Classe statique FilesManager 16](#_Toc484598774)

[Classe statique Globals 17](#_Toc484598775)

[Classe AppFile 17](#_Toc484598776)

[3.2.2 Interfaces 18](#_Toc484598777)

[3.2.3 Vérification du formulaire de connexion 20](#_Toc484598778)

[Vérification du nom du serveur IMAP 21](#_Toc484598779)

[Vérification du nom du numéro de port 21](#_Toc484598780)

[Vérification du mail 21](#_Toc484598781)

[Vérification du mot de passe 21](#_Toc484598782)

[Vérification du dossier racine 22](#_Toc484598783)

[3.2.4 Connexion IMAP 22](#_Toc484598784)

[3.2.5 Interactions base de données 23](#_Toc484598785)

[3.2.6 Détection des modifications de fichiers 24](#_Toc484598786)

[3.2.7 Organisation des fichiers dans la boite mail 25](#_Toc484598787)

[3.2.8 Transformation des fichiers « dossier – boite mail » 25](#_Toc484598788)

[3.2.9 La synchronisation des fichiers 26](#_Toc484598789)

[Les synchronisations initiales et au démarrage 27](#_Toc484598790)

[3.2.10 Diagramme de flux du fonctionnement de l’application 28](#_Toc484598791)

[3.3 Conception des tests 29](#_Toc484598792)

[3.3.1 Scénario N°1 : Connexion 29](#_Toc484598793)

[3.3.2 Scénario N°2 : Synchronisation 29](#_Toc484598794)

[3.3.3 Scénario N°3 : Changement de compte 30](#_Toc484598795)

[3.3.4 Scénario N°4 : Espace disponible 30](#_Toc484598796)

[3.3.5 Scénario N°5 : Synchronisation initiale 30](#_Toc484598797)

[3.3.6 Scénario N°6 : Création de fichiers 31](#_Toc484598798)

[3.3.7 Scénario N°7 : Suppression de mail 31](#_Toc484598799)

[3.3.8 Scénario N°8 : Suppression de fichier 31](#_Toc484598800)

[3.3.9 Scénario N°9 : Édition de fichier 32](#_Toc484598801)

[3.3.10 Scénario N°10 : Gestion des erreurs 32](#_Toc484598802)

[3.4 Planification détaillée 33](#_Toc484598803)

[3.4.1 Semaine 1 33](#_Toc484598804)

[3.4.2 Semaine 2 34](#_Toc484598805)

[3.4.3 Semaine 3 35](#_Toc484598806)

[3.4.4 Semaine 4 36](#_Toc484598807)

[3.4.5 Semaine 5 37](#_Toc484598808)

[3.4.6 Semaine 6 38](#_Toc484598809)

[4 Réalisation 40](#_Toc484598810)

[4.1 Dossier de Réalisation 40](#_Toc484598811)

[4.1.1 Interface utilisateur 40](#_Toc484598812)

[Fenêtre de login 40](#_Toc484598813)

[Fenêtre de traitement des fichiers 44](#_Toc484598814)

[4.1.2 Connexion au serveur IMAP et interactions boite mail 48](#_Toc484598815)

[4.1.3 Interactions avec la base de données 52](#_Toc484598816)

[4.1.4 La synchronisation automatique des fichiers 54](#_Toc484598817)

[Mise à jour de la liste locale 59](#_Toc484598818)

[Mise à jour de la liste distante 60](#_Toc484598819)

[Envoi des nouveaux fichiers sur la boite mail 61](#_Toc484598820)

[Suppression des fichiers de la boite mail 62](#_Toc484598821)

[4.1.5 Synchronisation initiale 64](#_Toc484598822)

[4.1.6 Synchronisation au démarrage 66](#_Toc484598823)

[4.1.7 Mail d’index des dossiers 67](#_Toc484598824)

[4.1.8 Cryptage du mot de passe de l’utilisateur 68](#_Toc484598825)

[4.2 Dossier des tests 69](#_Toc484598826)

[4.2.1 Scénario N°1 : Connexion 69](#_Toc484598827)

[4.2.2 Scénario N°2 : Synchronisation 69](#_Toc484598828)

[4.2.3 Scénario N°3 : Changement de compte 70](#_Toc484598829)

[4.2.4 Scénario N°4 : Espace disponible 70](#_Toc484598830)

[4.2.5 Scénario N°5 : Synchronisation initiale 71](#_Toc484598831)

[4.2.6 Scénario N°6 : Création de fichiers 71](#_Toc484598832)

[4.2.7 Scénario N°7 : Suppression de mail 72](#_Toc484598833)

[4.2.8 Scénario N°8 : Suppression de fichier 72](#_Toc484598834)

[4.2.9 Scénario N°9 : Édition de fichier 73](#_Toc484598835)

[4.2.10 Scénario N°10 : Gestion des erreurs 73](#_Toc484598836)

[4.2.11 Autres tests 74](#_Toc484598837)

[5 Conclusion 76](#_Toc484598838)

[5.1 Bilan des fonctionnalités demandées 76](#_Toc484598839)

[5.2 Fonctionnalités supplémentaires à implémenter 77](#_Toc484598840)

[5.2.1 Gestion des fichiers de grande taille 77](#_Toc484598841)

[5.2.2 Exécution simultanée sur plusieurs ordinateurs 78](#_Toc484598842)

[5.3 Bilan personnel 79](#_Toc484598843)

[6 Divers 82](#_Toc484598844)

[6.1 Journal de travail 82](#_Toc484598845)

[6.1.1 Semaine 1 82](#_Toc484598846)

[6.1.2 Semaine 2 83](#_Toc484598847)

[6.1.3 Semaine 3 84](#_Toc484598848)

[6.1.4 Semaine 4 85](#_Toc484598849)

[6.1.5 Semaine 5 86](#_Toc484598850)

[6.1.6 Semaine 6 87](#_Toc484598851)

[6.2 Gantt final 88](#_Toc484598852)

[6.2.1 Semaine 1 88](#_Toc484598853)

[6.2.2 Semaine 2 89](#_Toc484598854)

[6.2.3 Semaine 3 90](#_Toc484598855)

[6.2.4 Semaine 4 91](#_Toc484598856)

[6.2.5 Semaine 5 92](#_Toc484598857)

[6.2.6 Semaine 6 93](#_Toc484598858)

[6.3 Webographie 94](#_Toc484598859)

[6.4 Cahier des charges 95](#_Toc484598860)

Spécifications

# Spécifications[[1]](#footnote-1)

## Titre

MailStorage

## Description

MailStorage est une application qui permet d’utiliser l’espace disponible sur une boite mail pour stocker des fichiers. Au démarrage un dossier racine est spécifié et l’utilisateur peut y prendre ou placer des fichiers. Le tout est ensuite synchronisé dans la boite mail ce qui permet d’accéder aux fichiers depuis n’importe quel ordinateur ayant une connexion Internet.

L’avantage d’une telle solution permet de disposer d’une petite espace de stockage et de pouvoir y accéder depuis presque n’importe quel ordinateur. Cela offre aussi un avantage : il n’est pas nécessaire de créer un compte pour bénéficier de cet espace de stockage. En effet, une simple adresse mail suffit amplement.

## Matériel et logiciels à disposition

Ci-dessous la liste du matériel à disposition pour le projet :

|  |
| --- |
| Ordinateur ETML |
| MacBook Pro personnel |
| Microsoft Office |
| Libre Office |
| Visual Studio Community 2017 |
| ReSharper Ultimate |
| Un accès à Internet |
| Des modèles de fichiers pour la documentation |

## Prérequis

La réalisation de ce projet nécessite d’avoir acquis les compétences en programmation C# structurée, événementielle, orientée-objet et en gestion de projet. Toutes ces compétences ont été acquises lors des cours à l’ETML.

Planification initiale

# Planification Initiale

La planification initiale consiste à définir une liste de tâches représentant les grands thèmes du projet afin d’obtenir une vision d’ensemble. La planification permet ainsi d’évaluer les délais et les difficultés techniques du travail à fournir. Un diagramme de Gantt est aussi créé afin de donner un aspect visuel à la charge de travail.

## Liste des tâches

|  |  |
| --- | --- |
| N° | Description de la tâche |
| 1 | Découverte du projet et lecture du cahier des charges |
| 2 | Mise en place de l’environnement de travail |
| 3 | Planification des tâches |
| 4 | Analyse technique |
| 5 | Réalisation et implémentation des fonctionnalités |
| 6 | Test des fonctionnalités et résolution des bugs |
| 7 | Documentation du projet |

## Diagramme de Gantt

Les tâches sont réparties sur les 5.5 semaines de travail. Chaque semaine ne compte que 4 jours de travail consacrés au TPI. La planification détaillée qui se trouve dans l’analyse est basée sur cette planification initiale.

# 

Analyse

# Analyse

L’analyse est l’une des phases les plus importantes du projet, cette étape permet d’en éclaircir chaque partie technique, de décider de quelle façon le développement et la logique seront organisés et de trouver des solutions aux éventuelles difficultés algorithmiques ou technologiques.

## Opportunités

### Utilisation du protocole IMAP[[2]](#footnote-2)

Une des composantes principales de ce projet est l’utilisation du protocole IMAP pour stocker et récupérer des fichiers depuis une boite mail. Ce protocole représente un défi technique et doit être mis en place correctement afin d’éviter des erreurs de connexion ou d’authentification.

Envoi et réception de données avec le protocole IMAP

Programme

MailStorage

Boite mail

Le protocole IMAP, Internet Message Access Protocol, est l’un des protocoles utilisés par les services mail. Son utilité est de pouvoir récupérer des mails en se connectant sur une boite mail.

Il existe deux protocoles permettant la récupération de mails : l’IMAP et le POP (Post Office Protocol). L’avantage de l’IMAP est qu’il permet la gestion des dossiers dans la boite mail, contrairement au protocole POP qui ne permet que de récupérer la liste des mails présents dans la boite de réception.[[3]](#footnote-3)

Créé en 1986, le protocole IMAP n’a cessé d’évoluer et en est actuellement à sa version 4.

#### Solution

L’implémentation du protocole IMAP dans le projet peut se faire via la librairie MailKit. Cet API permet de créer des connexions IMAP4, POP3 et SMTP ver un serveur mail spécifié en C#. Idéal pour l’envoi et la réception de mails.

### Stockage de fichiers sur boite mail

L’utilité primaire d’une boite mail consiste à recevoir des messages d’autres personnes. Ces messages contiennent généralement un sujet, un contenu, l’adresse mail de l’émetteur, l’adresse du destinataire et parfois, le mail peut aussi être accompagné d’une pièce jointe.

Mail

**De :** [personne@mail.com](mailto:personne@mail.com)

**À :** [moi@mail.com](mailto:moi@mail.com)

**Sujet :** Titre du message

**Contenu :** Texte du message

Cependant, il n’est pas impossible d’utiliser ces messages pour y joindre d’autres éléments, il n’y a pas de restrictions sur le texte présent dans le sujet et le contenu mit à part la taille du mail.

Dans ce projet, le but est de pouvoir exploiter la boite mail de l’utilisateur afin d’y stocker des fichiers et leur arborescence. Cela peut être accompli de différentes façons.

#### Solution

Le but de ce programme est de pouvoir envoyer des fichiers sur la boite mail afin de pouvoir y accéder depuis n’importe quel ordinateur.

Les fichiers une fois identifiés peuvent être convertis en binaire, puis en texte grâce à une fonction de conversion en Base64. La conversion du code binaire en Base64 transforme tous les « 1 et 0 » en caractères ASCII. Comme un caractère ASCII nécessite un groupe de 6 bits, cela permet d’avoir une chaine de caractères sortante bien plus petite que si elle était en binaire.

Fichier en données Base64

Fichier en données binaires

01010011010101010100110101011010101101010110010101101101

1010101101010110010101101101

Fichier

Windows

5pWt

Une fois le fichier transcrit en texte Base64, il peut être stocké dans le contenu d’un mail. Les autres informations du fichier telles que le nom, le chemin d’accès et les dates de création et de modification peuvent être stockées dans le sujet du message.

Fichier en Base64

Mail

5pWt

**De :** [mailstorage@mail.com](mailto:mailstorage@mail.com)

**À :** [moi@mail.com](mailto:moi@mail.com)

**Sujet :** C:\Users\name\file.txt

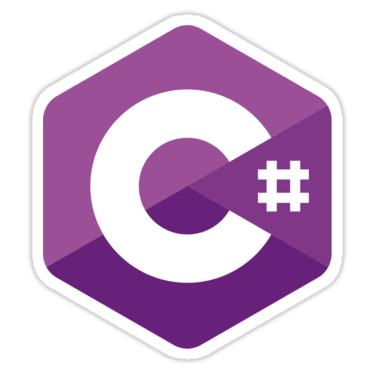
**Contenu :** 5pWt

Pour la récupération des fichiers, il suffit simplement de télécharger le mail depuis le serveur IMAP, transcrire le contenu Base64 en fichier Windows et lui appliquer son nom ainsi que les dates de création et de modification.

### Approfondissement des connaissances

Durant ce projet certaines compétences seront approfondies :

* Compétences en développement C#
* Développement orienté-objet
* Utilisation de la librairie SQLite
* Amélioration de la gestion de projet



## 

## Document d’analyse et conception

### Architecture du projet

Le projet doit être capable d’accomplir certaines tâches et pour se faire, des classes spécifiques sont nécessaires :

* Une classe pour la fenêtre de connexion
* Une classe pour la fenêtre principale
* Une classe statique pour les variables globales
* Une classe statique pour communiquer en IMAP avec la boite mail
* Une classe pour communiquer avec la base de données
* Une classe pour gérer les fichiers
* Une classe représentant un fichier

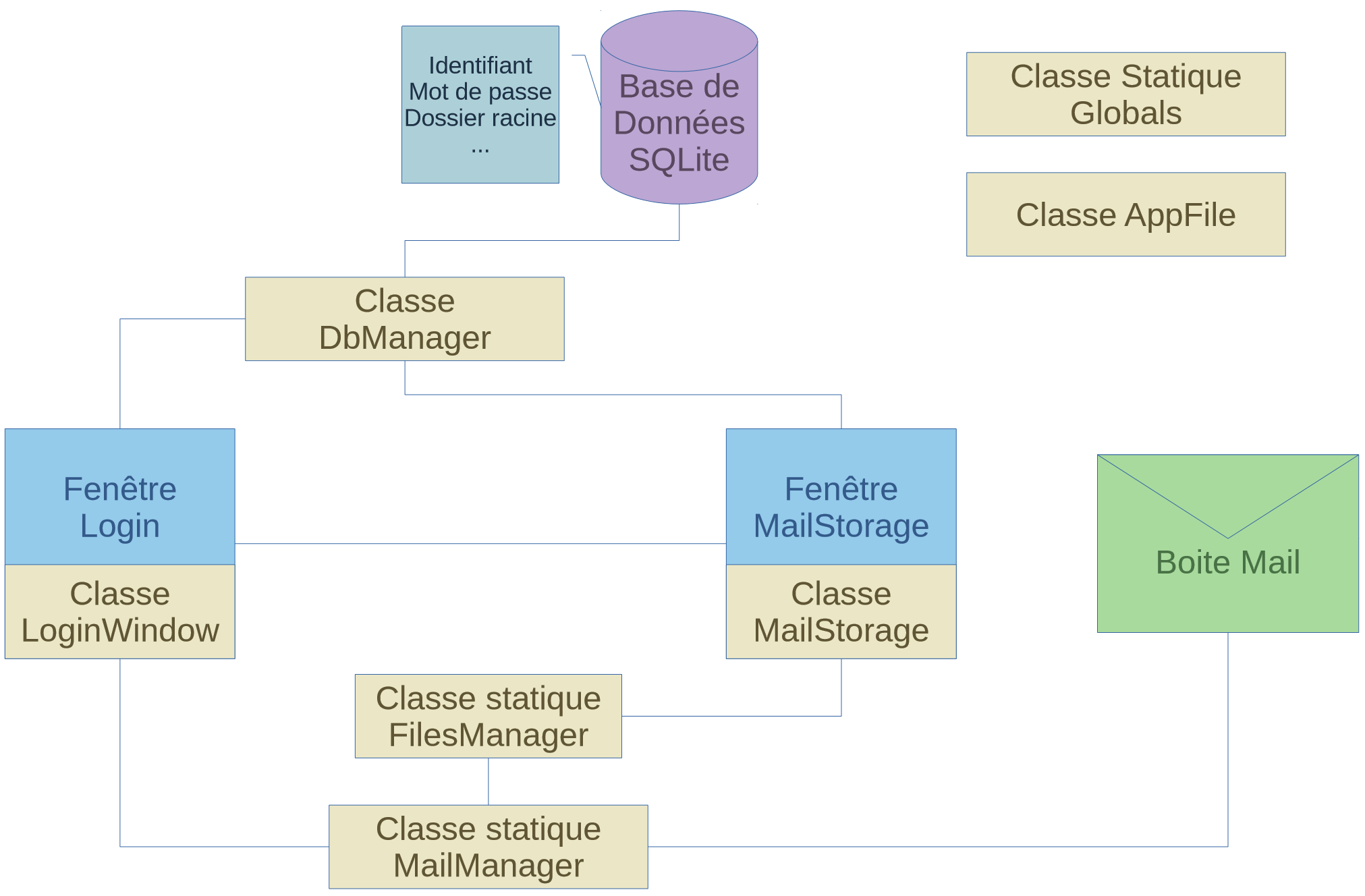
Les classes principales sont celles qui gèrent les fenêtres, c’est depuis ces éléments que les autres classes sont instanciées et appelées.

Figure 1 : Schéma des classes et des éléments de l’application

#### Classe LoginWindow

Cette classe représente la fenêtre de login, c’est dans ce code que se font les interactions avec la base de données. Les valeurs du formulaire de connexion présent dans la fenêtre sont ajoutées à la base de données si elle est vide. Si la base de données contient déjà des valeurs, celles-ci sont remplacées par les nouvelles.

#### Classe DbManager

Toutes les interactions directes avec la base de données sont faites dans cette classe. Les fonctions disponibles permettent d’ajouter et de retirer des données.

#### Classe MailStorage

La fenêtre principale est gérée par la classe *MailStorage*. C’est dans ce code que la vérification continuelle des fichiers s’exécute. Les fichiers ajoutés au dossier local sont envoyés à la boite mail et les fichiers manquants dans le dossier local y sont rajoutés via la classe *FileManager*.

#### Classe statique MailManager

Pour envoyer et recevoir des mails, il faut une connexion IMAP constante. C’est le rôle de la classe *MailManager*. Les envois et réceptions de mail ainsi que leur contenu y est géré.

Au lancement du programme, une connexion IMAP est ouverte avec les données de la base de données afin de pouvoir prendre le contrôle de la boite mail.

#### Classe statique FilesManager

Dans cette classe, les fichiers sont transformés en mails ou inversement. Cela permet d’obtenir les données dans la forme souhaitée.

Les fichiers sont transformés en texte Base64 qui sont ensuite placés dans le contenu du mail, tandis que les mails récupérés sont retransformés en fichiers afin d’être placés dans le dossier local.

C’est depuis cette classe que l’utilisation de la classe *MailManager* se fait. *FilesManager* se charge d’envoyer les données du fichier pour que l’envoi dans la boite mail se fasse.

#### Classe statique Globals

Toutes les variables importantes qui doivent être accessibles partout dans le code se trouvent dans la classe statique *Globals*.

#### Classe AppFile

Cette classe représente un fichier. Elle est utilisée afin de disposer de tous les attributs des fichiers : nom, date de création, date de modification, chemin d’accès, etc.

Les instances de cette classe représentent les fichiers du dossier local et de la boite mail. Elles sont stockées dans les listes qui répertorient tous les fichiers.

### Interfaces

Le programme ne comporte que deux fenêtres car la majorité des manipulations se font dans l’explorateur Windows. La fenêtre principale est conçue pour tourner en arrière-plan afin que les fichiers se mettent à jour automatiquement.

La première fenêtre s’affichant au démarrage de l’application est celle de connexion.

C’est dans cette fenêtre que les champs de configuration du serveur IMAP, d’adresse mail, de mot de passe et de dossier racine sont spécifiés.

Une action sur le bouton suivant lance la tentative de connexion IMAP avec la boite mail.

Figure 2 : Fenêtre de connexion

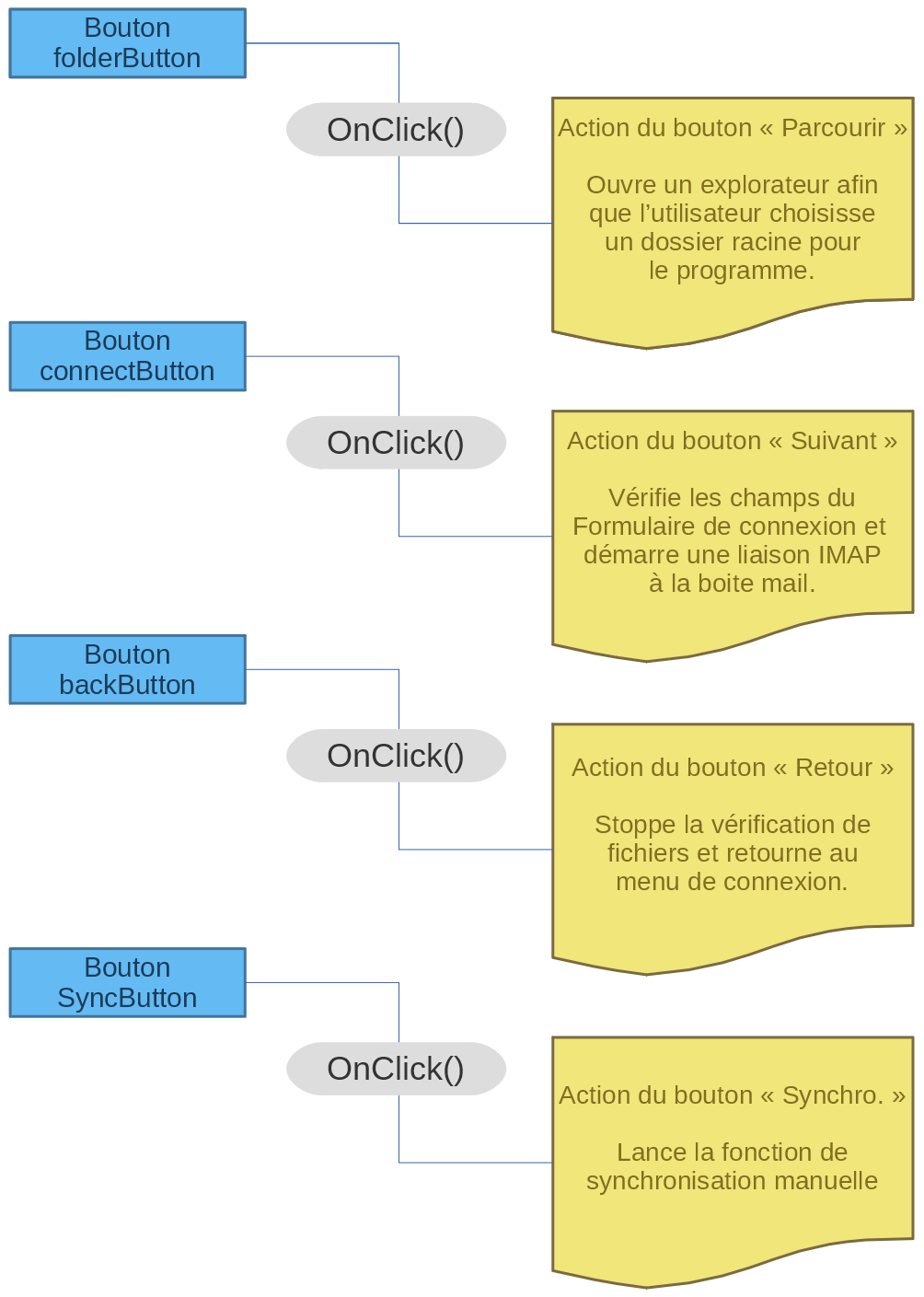


La deuxième fenêtre représente la connexion active de l’application.

Les seules actions possibles sont le retour au login et la fonction de synchronisation manuelle.

Le statut de la connexion est affiché afin de toujours avoir une confirmation du fonctionnement du programme.

Figure 3 : Fenêtre de traitement des fichiers

Les interactions avec l’interface ne sont pas très nombreuses. Ce ne sont que des boutons qui réagissent au clic. Le schéma événementiel ci-dessous décrit les interactions possibles avec l’interface et les actions que cela produit.

Sur la première fenêtre de connexion, deux boutons sont présents : le bouton de sélection du dossier racine et le bouton de connexion à la boite mail.

Sur la deuxième fenêtre, les boutons disponibles sont : le bouton de retour à la fenêtre de connexion et le bouton servant à lancer une synchronisation IMAP manuelle.

### Vérification du formulaire de connexion

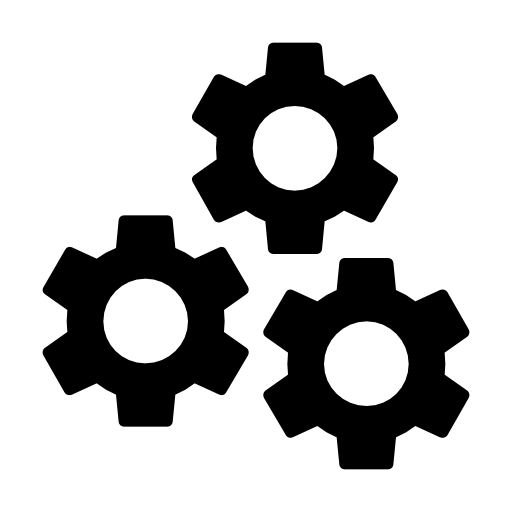
La première étape lors de l’utilisation du programme est la connexion en IMAP au serveur mail. Pour ce faire, l’utilisateur doit entrer plusieurs données dans un petit formulaire afin que le programme ait les valeurs nécessaires à la connexion.



Pour commencer, l’utilisateur entre le nom du serveur IMAP ainsi que le port correspondant.

Ensuite, il entre son adresse mail, le mot de passe de son compte mail et sélectionne le dossier racine ou il veut gérer ses fichiers.

Enfin, la confirmation se fait en cliquant sur le bouton « Suivant ».

Les valeurs sont ensuite vérifiées dans le programme afin de s’assurer que l’utilisateur a bien entré des données valides. Dans le cas où la validation retourne une erreur, un message d’avertissement s’affiche à l’écran pour avertir l’utilisateur.

Erreur :

Le mail entré n’est pas valide. Veuillez en spécifier un autre.

Valide :

Début de la connexion IMAP au serveur mail.

adresse@mail.com

#### Vérification du nom du serveur IMAP

Le nom du serveur IMAP ne possède pas beaucoup de vérifications. Le programme vérifie juste que le champ de formulaire n’est pas vide. Dans le cas contraire, un message d’erreur est affiché.

Généralement un nom de serveur IMAP est composé de trois ou quatre parties. Par exemple, le nom du serveur IMAP de Gmail est « imap.gmail.com ».

#### Vérification du nom du numéro de port

Comme pour la vérification du nom du serveur IMAP, celle pour le numéro de port ne vérifie que si le champ de formulaire n’est pas vide et affiche un message d’erreur si tel est le cas.

#### Vérification du mail

Pour être valide, une adresse mail doit respecter plusieurs conditions :

* Posséder un symbole « @ »
* Au moins 2 caractères avant le symbole « @ »
* Au moins 2 caractères après le symbole « @ »
* Un caractère « . » après le symbole « @ »
* Une extension après le caractère « . »

adresse@mail.com

#### Vérification du mot de passe

Aucune vérification particulière n’est effectuée sur le mot de passe si ce n’est que le champ de formulaire ne doit pas être vide.

Le mot de passe spécifié doit correspondre au compte mail afin que le programme puisse ensuite s’authentifier auprès du serveur IMAP et puisse lancer la connexion qui permettra de transférer les fichiers.

#### Vérification du dossier racine

La première vérification et s’assure simplement que le la valeur n’est pas vide et qu’elle contient du texte.

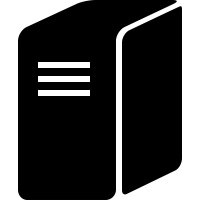
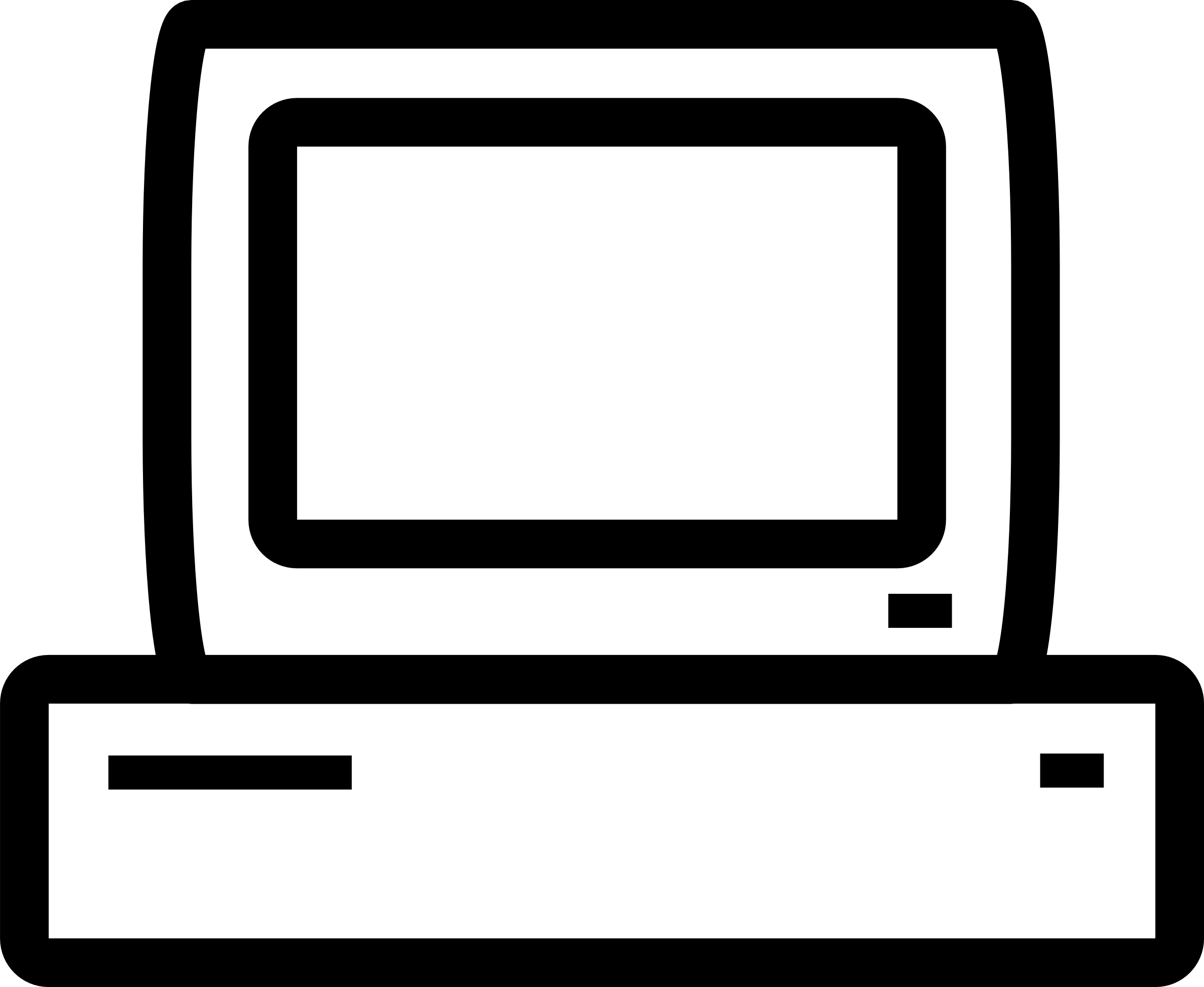
Dans ce champ, l’utilisateur ne peut rentrer des données manuellement. Pour remplir le champ il faut cliquer sur le bouton « Parcourir » afin qu’une fenêtre d’explorateur Windows s’ouvre pour que l’utilisateur puisse choisir le dossier racine qu’il souhaite.

L’ouverture d’un explorateur Windows est effectuée grâce à la classe *FolderBrowserDialog* disponible dans le namespace *System.Widows.Forms*.[[4]](#footnote-4)

Le chemin du dossier est ensuite récupéré puis stocké dans la base de données.

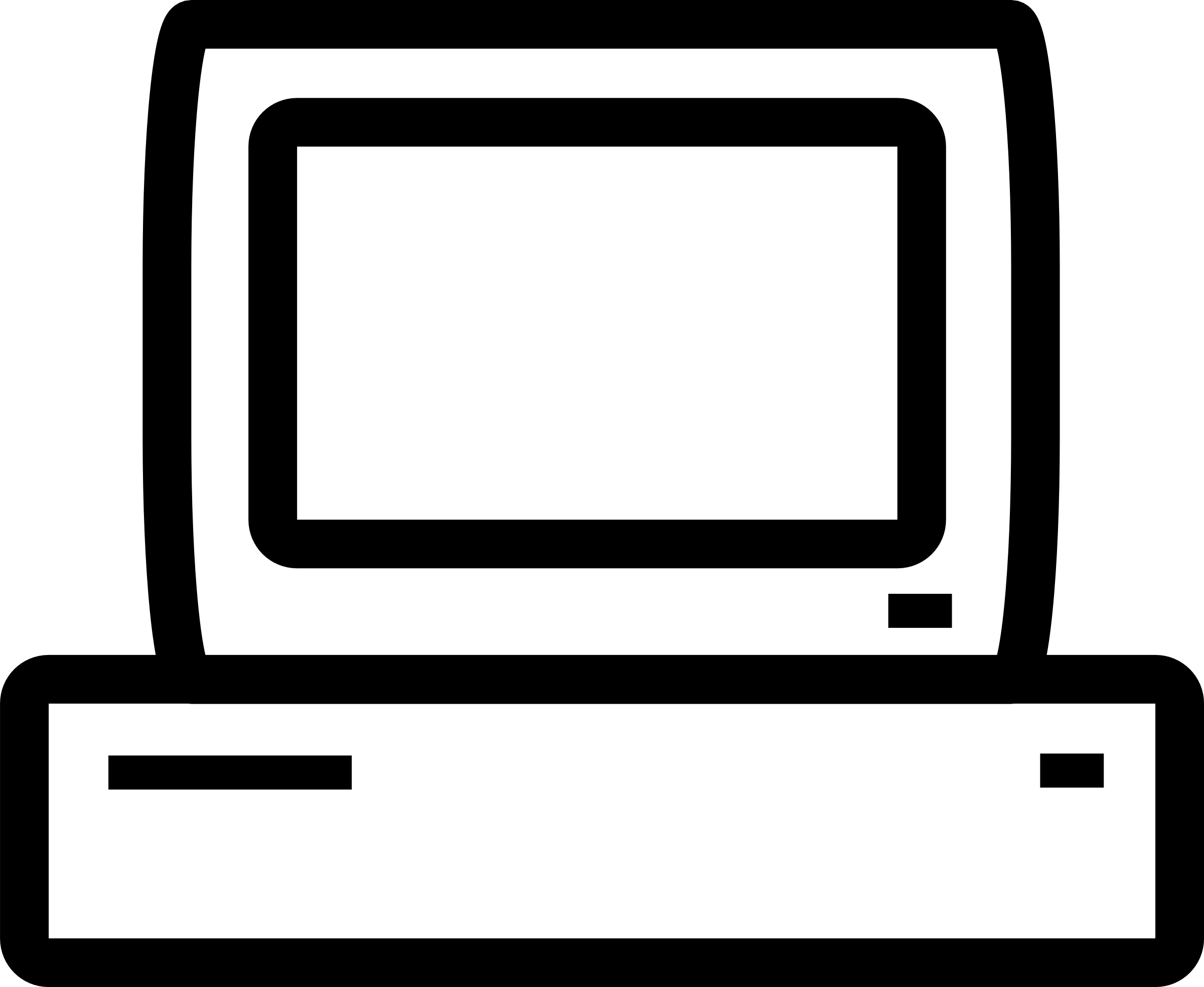
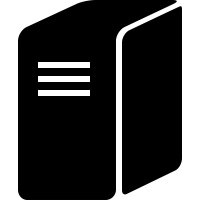
### Connexion IMAP

Une fois les données validées, le programme doit tenter de se connecter au serveur mail. La classe *MailManager* est instanciée afin de disposer des moyens nécessaires à la connexion. La librairie utilisée est MailKit. Elle permet de communiquer en SMTP, POP3 ou IMAP4 avec les serveurs mail.

 Tout d’abord le programme commence par établir une connexion sur le serveur mail, il s’annonce en spécifiant le nom du serveur, le port à utiliser et si la connexion est du type SSL. Dans la plupart des serveurs, le port IMAP utilisé est le 993 pour une connexion SSL.

Connexion OK

Imap.gmail.com, 993, SSL

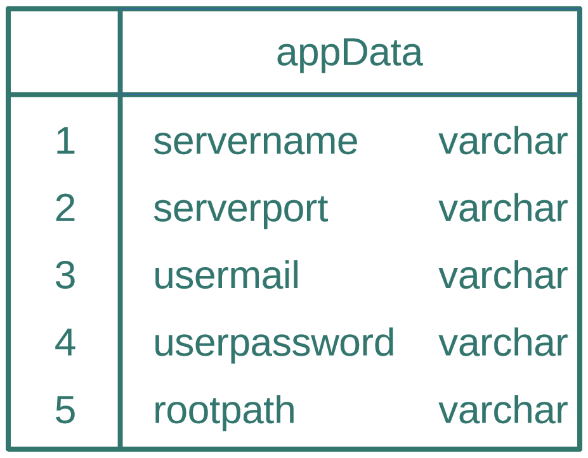
Ensuite, une demande d’authentification est lancée avec les identifiants de l’utilisateur (adresse mail et mot de passe). Le serveur répond soit avec un OK soit avec un refus.

User OK, Auth OK

[user@gmail.com](mailto:user@gmail.com), \*\*\*\*\*\*\*\*\*

Lorsque le programme est connecté et authentifié, il peut accéder, créer ou supprimer des mails dans la boite mail de l’utilisateur. La connexion IMAP est établie.

### Interactions base de données

La base de données utilisée dans ce programme est très petite, elle ne comporte qu’une seule table de données.

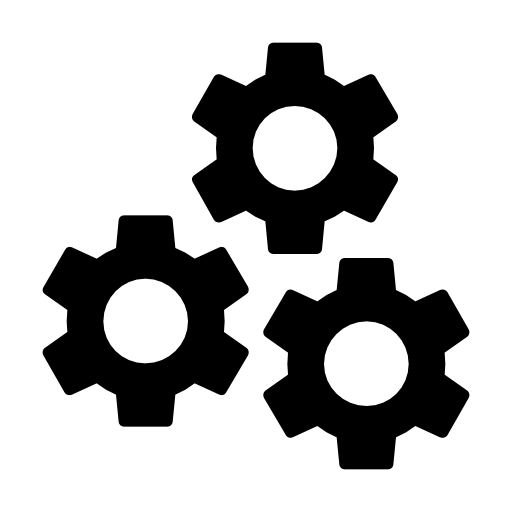
La table appData est composée de 5 champs. Tout d’abord, les deux champs concernant le nom et le port du serveur IMAP.

Ensuite viennent les champs de l’utilisateur qui stockent son adresse mail et son mot de passe crypté ainsi que le dossier racine choisi.

L’utilisation d’une base de données pour si peu de valeurs se justifie par le fait que cela permet l’ajout d’autres fonctionnalités si le temps le permet (la gestion de plusieurs utilisateurs par exemple) ou si le projet est ensuite réutilisé par une autre personne.

En effet, un simple fichier texte aurait suffi pour réaliser le travail demandé. Cependant, n’ayant jamais utilisé de base de données SQLite, ce projet est un bon moyen de découvrir le fonctionnement de cette librairie.

Les interactions avec la base de données se font via la classe *DbManager*. C’est le seul endroit du code qui va directement interagir avec les données stockées dans la base de données pour les retransmettre ou les insérer.

L’API SQLite est une librairie qui sert d’intermédiaire entre le code C# et la base de données. Les requêtes sur la base de données sont écrites en SQL, ce qui permet de posséder toutes les fonctionnalités d’une base de données intégrée à un programme C#.

SELECT \* FROM table

API SQLite

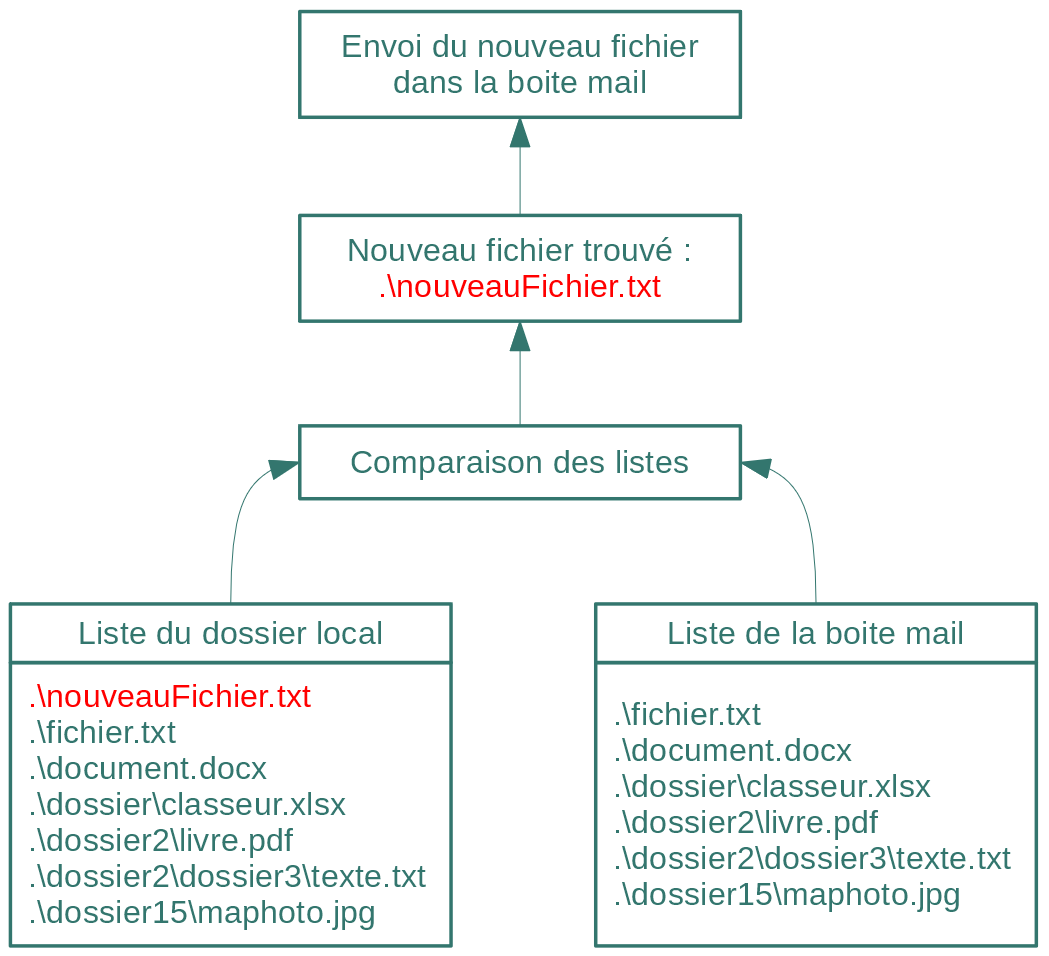
Base de données

Programme

### Détection des modifications de fichiers

La détection des changements dans les fichiers de l’arborescence est l’une des fonctionnalités principales du programme. C’est grâce à cette détection que les fichiers sont placés et mis à jour en même temps dans le dossier local et la boite mail de l’utilisateur.

La détection des fichiers fonctionne à base de listes de fichiers. Lorsqu’un fichier est ajouté, soit dans le dossier local, soit dans la boite mail, les listes sont comparées et l’élément manquant est placé dans le dossier ou la boite mail.

Lors de l’exécution du programme, deux listes sont actives en permanence : la liste des fichiers du dossier local et la liste des fichiers présents dans la boite mail. Ces listes contiennent les chemins d’accès aux fichiers et leur date de modification, ce qui permet de connaitre l’arborescence totale du dossier racine ainsi que le moment ou un fichier est modifié, supprimé, renommé ou ajouté.

Lorsqu’un fichier est ajouté dans le dossier racine (voir illustration ci-dessus), la liste va récupérer son chemin et lorsque la comparaison des deux listes sera effectuée, le fichier sera indiqué comme nouveau et sera envoyé sur la boite mail afin que les deux listes s’égalisent et continuent à posséder les mêmes valeurs.

### Organisation des fichiers dans la boite mail

Afin de ne pas désorganiser complètement la boite mail de l’utilisateur, l’application crée un dossier « MailStorage » dans la boite mail où tous les fichiers sont déposés.

**Mail**

**Sujet**

C\...\...\...\calc.xlsx

**Contenu**

IdOssiaDoe9d8D9eDa

**Boite de réception**

Important

Spam

Corbeille

**Corbeille**

**Spam**

Corbeille

**Important**

Spam

Corbeille

**MailStorage**

**Sujet**

C\...\...\...\doc.docx

**Contenu**

ajfoiOEjfie97cs89es8w

**Mail**

**Sujet**

C\...\...\...\file.txt

**Contenu**

Zsri47d8qw9Rt8

**Mail**

Les fichiers déposés dans le dossier mail « MailStorage » sont tous composés de la même façon : le sujet du mail contient les informations du fichier dans le dossier local (nom, chemin d’accès, date de création et de modification) et le contenu contient le fichier complet sous forme de texte.

### Transformation des fichiers « dossier – boite mail »

Les fichiers qui se trouvent dans le dossier local et qui doivent être placés dans la boite mail sont récupérés via la comparaison des listes (voir section sur la détection des modifications de fichiers).

Une fois ces fichiers récupérés, ils sont transformés en Base64. La transformation en Base64 permet d’obtenir une chaine de caractères correspondant aux données binaires du fichier.

Fichier en données Base64

Fichier en données binaires

Fichier

Windows

01010011010101010100110101011010101101010110010101101101

1010101101010110010101101101

5pWt

Ce texte est ensuite placé dans la boite mail, dans le contenu d’un nouveau mail situé dans le dossier « MailStorage ». Les informations du fichier sont quant à elles placées dans le sujet du mail sous cette forme :

Sujet du mail

.\dossier\fichier.txt

Fichier.txt

::

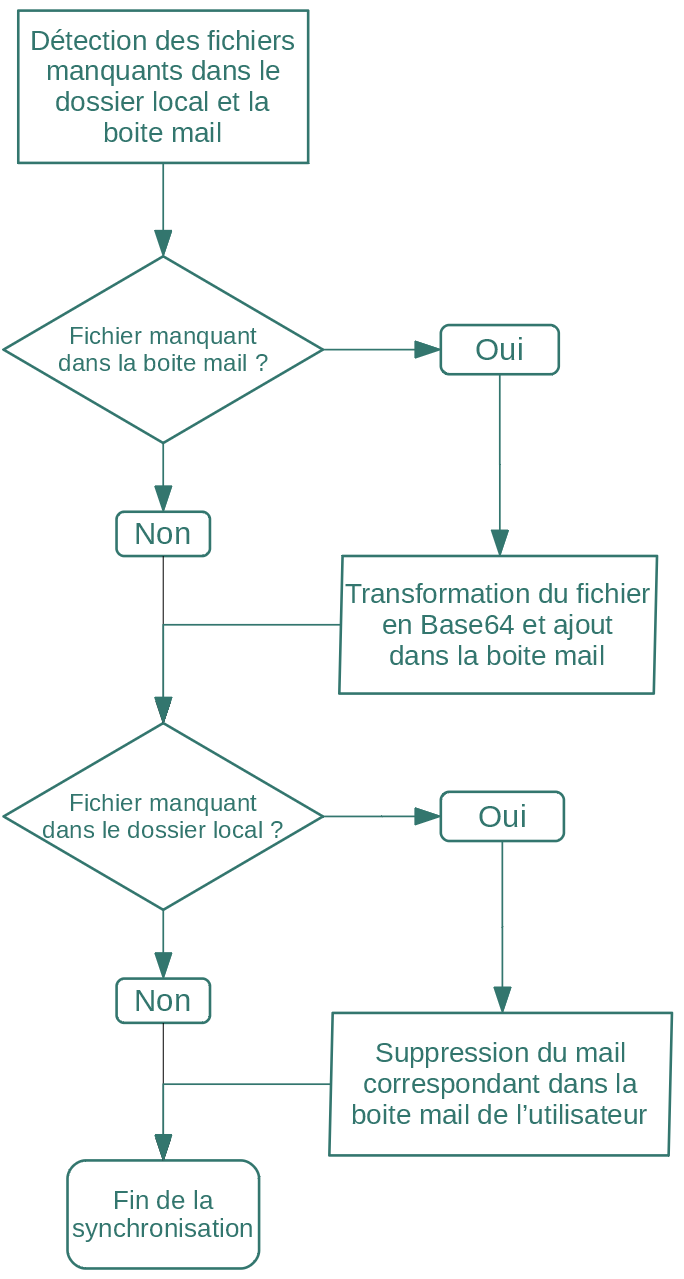
::

13.06.17

::

15.06.17

### La synchronisation des fichiers

Le rôle de la fonction de synchronisation consiste à reprendre tous les éléments du processus de transformation et de détection des fichiers.

Premièrement, la détection des fichiers manquants dans le dossier local et la boite mail s’effectue. Cela permet d’identifier quels fichiers il faut déplacer.

Ensuite, s’il manque des fichiers dans la boite mail, ces fichiers sont récupérés dans le dossier local, puis un par un, ils sont transformés en texte Base64 puis placés dans un mail dans la boite mail.

Dans le cas où un fichier a été supprimé du dossier local, sa suppression est répliquée dans la boite mail. Le mail correspondant au fichier est donc supprimé.

Lors de l’exécution du programme, ce sont les manipulations du dossier local qui ont la priorité sur la boite mail.

Figure 4 : Diagramme de flux du fonctionnement de la synchronisation

#### Les synchronisations initiales et au démarrage

Les synchronisations dites « initiales » et « au démarrage » sont deux types de synchronisations qui diffèrent de celle par défaut. En effet, ces deux processus vont donner la priorité à la boite mail plutôt qu’au dossier local. Cela permet de mettre à jour l’état du dossier racine lorsque celui-ci vient d’être spécifié ou que son état n’est plus à jour.

Les deux listes de fichiers sont donc comparées mais les fichiers qui sont dans la boite mail et absents du dossier local sont téléchargés tandis que les fichiers absents de la boite mail et présents dans le dossier local sont supprimés.

Cela permet à l’application d’être utilisée sur différents ordinateurs en ayant toujours les fichiers ajoutés sur d’autres ordinateurs à disposition.

Figure 5 : Diagramme de flux montrant le fonctionnement de la synchronisation initiale

### Diagramme de flux du fonctionnement de l’application

## Conception des tests

### Scénario N°1 : Connexion

Ce test consiste à fournir une adresse mail, un mot de passe et un dossier racine à l’application afin qu’elle établisse une connexion à la boite mail de l’utilisateur. La connexion doit se faire sans erreur et la fenêtre de traitement des fichiers doit s’afficher correctement.

|  |  |
| --- | --- |
| Étape | Description |
| Entrée | Fenêtre de connexion à l’application |
| Quand | Saisie du compte mail, do mot de passe et du dossier racine |
| Résultat | Connexion au compte mail et affichage de la fenêtre de traitement des fichiers |

### Scénario N°2 : Synchronisation

Le test de vérification du bouton synchroniser s’effectue afin de vérifier que la synchronisation « dossier local – boite mail » se fait sans erreur. Le résultat doit être visible lorsque la boite mail et le dossier racine possèdent les mêmes fichiers.

|  |  |
| --- | --- |
| Étape | Description |
| Entrée | Bouton de synchronisation manuelle des fichiers |
| Quand | Clic sur le bouton de synchronisation |
| Résultat | Mise à jour et synchronisation des fichiers « dossier - boite mail » |

### Scénario N°3 : Changement de compte

L’utilité de ce test est la vérification du bon fonctionnement de la base de données. Une fois un utilisateur enregistré, une autre adresse mail et mot de passe doivent être insérés dans le programme et la connexion doit se faire avec la nouvelle boite mail. Lors du prochain lancement de l’application, les nouveaux identifiants doivent se placer automatiquement dans les champs car ils auront été enregistrés dans la base de données.

|  |  |
| --- | --- |
| Étape | Description |
| Entrée | Fenêtre de connexion à l’application |
| Quand | Changement du compte mail, mot de passe et dossier racine |
| Résultat | Connexion au compte mail et affichage de la fenêtre de traitement des fichiers |

### Scénario N°4 : Espace disponible

L’affichage de l’espace disponible doit correspondre à l’espace réellement disponible dans la boite mail de l’utilisateur.

|  |  |
| --- | --- |
| Étape | Description |
| Entrée | Fenêtre de traitement des fichiers |
| Quand | À tout moment |
| Résultat | L’espace disponible affiché correspond à l’espace disponible dans la boite mail |

### Scénario N°5 : Synchronisation initiale

Après le lancement de l’application, les fichiers absents du dossier local doivent être ajoutés depuis les mails présents dans la boite mail de l’utilisateur. Au final, les fichiers du dossier local doivent être les mêmes que ceux de la boite mail.

|  |  |
| --- | --- |
| Étape | Description |
| Entrée | Dossier local |
| Quand | Après connexion à la boite mail |
| Résultat | Création automatique des fichiers depuis la boite mail |

### Scénario N°6 : Création de fichiers

Une fois l’application connectée à la boite mail. Tout fichier ajouté dans le dossier local doit être détecté par le programme et ajouté sous forme de mail dans la boite mail de l’utilisateur.

|  |  |
| --- | --- |
| Étape | Description |
| Entrée | Dossier local |
| Quand | Ajout d’un fichier dans le dossier |
| Résultat | Ajout automatique du fichier dans la boite mail |

### Scénario N°7 : Suppression de mail

Lorsqu’un mail de fichier est supprimé, le programme en cours d’exécution doit alors détecter la faille et recréer un nouveau mail avec le fichier concerné.

|  |  |
| --- | --- |
| Étape | Description |
| Entrée | Boite mail |
| Quand | Suppression d’un mail de fichier |
| Résultat | Recréation automatique du mail dans la boite mail |

### Scénario N°8 : Suppression de fichier

Lorsqu’un fichier est supprimé du dossier local, le programme doit également identifier le mail correspondant à ce fichier et le supprimer afin qu’il ne reste plus aucune trace du fichier.

|  |  |
| --- | --- |
| Étape | Description |
| Entrée | Dossier local |
| Quand | Suppression d’un fichier dans le dossier local |
| Résultat | Suppression automatique dans la boite mail |

### Scénario N°9 : Édition de fichier

Si un fichier est modifié depuis le dossier local, la nouvelle version doit venir remplacer l’ancienne dans la boite mail. L’ancienne version correspondante est identifiée et supprimée pour faire place à a nouvelle version.

|  |  |
| --- | --- |
| Étape | Description |
| Entrée | Dossier local |
| Quand | Édition d’un fichier dans le dossier local |
| Résultat | Mise à jour automatique du fichier dans la boite mail |

### Scénario N°10 : Gestion des erreurs

Si une fausse adresse mail ou un mauvais mot de passe sont entrés dans le formulaire de connexion, le programme doit être capable de gérer les exceptions et afficher un message d’erreur indiquant le problème à résoudre.

|  |  |
| --- | --- |
| Étape | Description |
| Entrée | Fenêtre de connexion à l’application |
| Quand | Insertion d’une fausse adresse mail et mauvais mot de passe |
| Résultat | Affichage des messages d’erreur |

## 

## Planification détaillée

### Semaine 1

|  |  |
| --- | --- |
| Tâches (chronologique) | Durée  [1/4h] |
| Lancement : Lecture du cahier des charges | 9 |
| Lancement : Création et rédaction de la planification initiale | 12 |
| Documentation : Rédaction du rapport de projet | 9 |
| Lancement : Mise en place de l’environnement | 9 |
| Analyse : Recherche des technologies utilisables | 6 |
| Analyse : Architecture du projet | 6 |
| Documentation : Rédaction du rapport de projet | 6 |
| Analyse : Fonctionnalité de connexion IMAP | 6 |
| Documentation : Rédaction du rapport de projet | 6 |
| Analyse : Détection des changements sur les fichiers | 9 |
| Documentation : Rédaction du rapport de projet | 21 |

### Semaine 2

|  |  |
| --- | --- |
| Tâches (chronologique) | Durée  [1/4h] |
| Analyse : Organisation des fichiers dans la boite mail | 6 |
| Documentation : Rédaction du rapport de projet | 6 |
| Analyse : Fonctionnalité de synchronisation | 12 |
| Documentation : Rédaction du rapport de projet | 6 |
| Analyse : Interface utilisateur | 3 |
| Documentation : Rédaction du rapport de projet | 3 |
| Test : Préparation des tests | 12 |
| Implémentation : Création de l’interface utilisateur | 12 |
| Implémentation : Connexion au serveur IMAP | 12 |
| Documentation : Rédaction du rapport de projet | 12 |
| Implémentation : Scan des fichiers modifiés | 15 |

### Semaine 3

|  |  |
| --- | --- |
| Tâches (chronologique) | Durée  [1/4h] |
| Documentation : Rédaction du rapport de projet | 12 |
| Implémentation : Ajout des fichiers modifiés à la boite mail | 18 |
| Documentation : Rédaction du rapport de projet | 9 |
| Implémentation : Téléchargement des fichiers plus récents en local | 18 |
| Documentation : Rédaction du rapport de projet | 9 |
| Test : Exécution des tests | 18 |
| Documentation : Rédaction du rapport de projet | 15 |

### Semaine 4

|  |  |
| --- | --- |
| Tâches (chronologique) | Durée  [1/4h] |
| Autres : Résolution de bug | 21 |
| Documentation : Rédaction du rapport de projet | 21 |
| Autres : Congé / Vacances / Examens | 57 |

### Semaine 5

|  |  |
| --- | --- |
| Tâches (chronologique) | Durée  [1/4h] |
| Autres : Congé / Vacances / Examens | 13 |
| Autres : Déploiement de l’application | 15 |
| Autres : Travail en plus | 21 |
| Documentation : Rédaction du rapport de projet | 50 |

### Semaine 6

|  |  |
| --- | --- |
| Tâches (chronologique) | Durée  [1/4h] |
| Autres : Congé / Vacances / Examens | 13 |
| Documentation : Rédaction du rapport de projet | 13 |

# 

Réalisation

# Réalisation

Dans cette section, toutes les étapes de la réalisation sont reprises et expliquées le plus techniquement possible. Chaque fonction est reprise et son fonctionnement expliqué, chaque difficulté technique rencontrée est détaillée dans le dossier de réalisation.

C’est aussi dans cette section que tous les tests sont effectués et retranscrits. Le dossier de tests contient tous les tests créés lors de l’analyse et réalisés sur le projet.

## Dossier de Réalisation

### Interface utilisateur

L’interface utilisateur possède deux fenêtres au total. La première représente le formulaire de connexion au serveur IMAP tandis que la seconde est la fenêtre de traitement automatique des fichiers.

#### Fenêtre de login

C’est la fenêtre qui possède le plus d’éléments à cause du formulaire de connexion.

TextBox : passwordTextBox

Label : passwordLabel

Label : titleLabel

Bouton : exitButton

Bouton : validateButton

Bouton : pathButton

TextBox : mailTextBox

TextBox : portTextBox

Label : portLabel

TextBox : pathTextBox

Label : pathLabel

Label : mailLabel

TextBox : serverTextBox

Label : serverLabel

Lors du lancement de l’application, la première fenêtre instanciée est la fenêtre de login. Cela permet à l’utilisateur d’entrer les dont le programme a besoin pour fonctionner correctement.

Une fois les valeurs entrées, une tentative de connexion à la boite mail est lancée. Dans le cas où la connexion réussit, les données fournies par l’utilisateur dans le formulaire de connexion sont stockées dans la base de données SQLite afin de connecter automatiquement l’utilisateur lors de la prochaine ouverture du programme.

Ci-dessous la liste complète de toutes les fonctions présentes dans la classe *LoginWindow*.

|  |  |
| --- | --- |
| Fonction | LoginWindow() - Constructeur |
| Paramètres | - |
| Retour | - |
| Description détaillée | LoginWindow() est le constructeur de la classe du même nom.  Dans cette fonction, les éléments de la fenêtre sont initialisés et affichés à l’écran.  La nouvelle instance s’ajoute ensuite à la classe Globals dans sa variable dédiée.  Cela permet d’accéder à cette instance de fenêtre de login depuis n’importe quel endroit du code. |

|  |  |
| --- | --- |
| Fonction | OnShown() - Événement |
| Paramètres | **Object** sender  **EventArgs** e |
| Retour | - |
| Description détaillée | Cette fonction est appelée lorsque tous les objets de la fenêtre sont tous affichés à l’écran.  Son but n’est que d’appeler la fonction FillFormWithExistingUser(). |

|  |  |
| --- | --- |
| Fonction | ValidateConnection() - Événement |
| Paramètres | **Object** sender  **EventArgs** e |
| Retour | - |
| Description détaillée | Ce code est appelé lorsque le bouton « Suivant » situé en dessous du formulaire est cliqué.  Premièrement le code commence par prendre chaque champ de formulaire et le valide. Si l’un des champs n’est pas valide, le booléen de vérification passe à « false ».  Ensuite, sir les champs sont valides, une connexion à la boite mail via la classe « MailManager » est lancée.  Si la connexion réussit, une instance de « DbManager » est créée et les données fournies par l’utilisateur.  Enfin, la fenêtre de login est cachée est celle de traitement des fichiers est instanciée. |

|  |  |
| --- | --- |
| Fonction | FillFormWithExistingUser() |
| Paramètres | - |
| Retour | - |
| Description détaillée | Cette fonction est systématiquement appelée lorsque le programme est lancé. C’est depuis la fonction « OnShow() » que l’appel est fait.  Dans ce code, une instance de connexion à la base de données est effectuée.  Une vérification est ensuite faite dans la base de données afin de récupérer des données existantes s’il y en a.  Dans le cas où il existe des données dans la base de données, elles sont récupérées et placées dans les champs de formulaire. La fonction « ValidateConnection() » est ensuite appelée.  Cela permet la connexion automatique de l’utilisateur si ses données sont enregistrées dans la base de données. |

|  |  |
| --- | --- |
| Fonction | SelectDirectory() - Événement |
| Paramètres | **Object** sender  **EventArgs** e |
| Retour | - |
| Description détaillée | Appelée quand le bouton « Parcourir » est cliqué.  Cette fonction affiche simplement une fenêtre d’explorateur afin que l’utilisateur choisisse son dossier racine. |

|  |  |
| --- | --- |
| Fonction | MoveWindow() - Événement |
| Paramètres | **Object** sender  **EventArgs** e |
| Retour | - |
| Description détaillée | Fonction appelée lorsque l’utilisateur déplace la fenêtre. |

|  |  |
| --- | --- |
| Fonction | ExitApplication() - Événement |
| Paramètres | **Object** sender  **EventArgs** e |
| Retour | - |
| Description détaillée | Fonction appelée lorsque l’utilisateur clique sur le bouton pour quitter l’application. |

#### Fenêtre de traitement des fichiers

La fenêtre de traitement des fichiers possède moins d’éléments que celle de login par contre la plupart des exécutions de code se font en arrière-plan. Ci-dessous la description des éléments de la fenêtre de traitement de fichiers.

Label : spaceBackLabel

PictureBox : statusPictureBox

Bouton : syncButton

Label : spaceLabel

Bouton : backButton

Label : statusLabel

Label : titleLabel

Bouton : exitButton

Bouton : minimizeButton

Cette fenêtre se charge de surveiller constamment le dossier racine choisi par l’utilisateur. Dès qu’un fichier est créé, déplacé, modifié, renommé ou supprimé, l’appel de la fonction de synchronisation est effectué. Les informations sur l’état de l’application sont affichées dans les labels de statuts ainsi que dans le label d’espace disponible qui, comme son nom l’indique, affiche l’espace disponible restant sur la boite mail de l’utilisateur.

La fenêtre de gestion des fichiers correspond à la classe *MailStorage* décrite ci-dessous.

|  |  |
| --- | --- |
| Fonction | MailStorage() - Constructeur |
| Paramètres | - |
| Retour | - |
| Description détaillée | Constructeur de la classe. Initialise tous les éléments de la fenêtre et démarre la surveillance du dossier racine. |

|  |  |
| --- | --- |
| Fonction | InitializeWindow() |
| Paramètres | - |
| Retour | - |
| Description détaillée | Vérifie si l’utilisateur a choisi un nouveau dossier et si tel est le cas, démarre la synchronisation initiale avec la boite mail. Redémarre ensuite le monitoring du dossier si nécessaire. |

|  |  |
| --- | --- |
| Fonction | BackButtonClick() - Événement |
| Paramètres | **Object** sender  **EventArgs** e |
| Retour | - |
| Description détaillée | Déconnecte l’application du serveur IMAP et retourne sur la fenêtre de login dans le cas ou aucune synchronisation avec la boite mail n’est en cours. Stoppe la surveillance du dossier racine. |

|  |  |
| --- | --- |
| Fonction | SyncButton() - Événement |
| Paramètres | **Object** sender  **EventArgs** e |
| Retour | - |
| Description détaillée | Lance une synchronisation manuelle des fichiers en appelant la fonction « SynchroniseFiles() ». |

|  |  |
| --- | --- |
| Fonction | MoveWindow() - Événement |
| Paramètres | **Object** sender  **MouseEventArgs** e |
| Retour | - |
| Description détaillée | En cas d’appui prolongé sur la barre de déplacement de la fenêtre, effectue un déplacement de la fenêtre par rapport à la position de la souris. |

|  |  |
| --- | --- |
| Fonction | ExitApplication() - Événement |
| Paramètres | **Object** sender  **MouseEventArgs** e |
| Retour | - |
| Description détaillée | Appelé lors de l’appui sur le bouton de fermeture de l’application. Un message d’avertissement est affiché si l’utilisateur essaye de quitter l’application pendant une synchronisation. |

|  |  |
| --- | --- |
| Fonction | MinimizeWindow() - Événement |
| Paramètres | **Object** sender  **MouseEventArgs** e |
| Retour | - |
| Description détaillée | Appelée lors de l’appui sur le bouton de réduction de la fenêtre. Au clic, l’application se réduit dans la barre des programmes en arrière-plan et un message d’information avertissant de l’exécution de l’application en arrière-plan avertit l’utilisateur. |

|  |  |
| --- | --- |
| Fonction | MaximizeWindow() - Événement |
| Paramètres | **Object** sender  **EventArgs** e |
| Retour | - |
| Description détaillée | Affiche la fenêtre si elle est réduite. Appelée lors du clic sur l’icône de l’application dans la barre des programmes. |

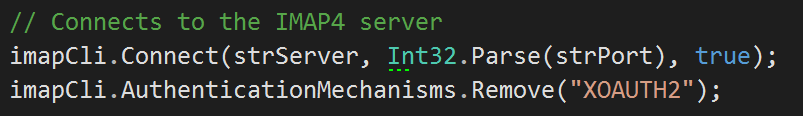
|  |  |
| --- | --- |
| Fonction | UpdateMailboxSpace() |
| Paramètres | - |
| Retour | - |
| Description détaillée | Récupère les informations de la boite mail de l’utilisateur et calcule l’espace restant pour l’afficher à l’écran. |

|  |  |
| --- | --- |
| Fonction | UpdateCurrentFile() |
| Paramètres | **String** strText |
| Retour | - |
| Description détaillée | Met à jour le label d’information sur le fichier en cours de traitement avec le texte passé en paramètres. |

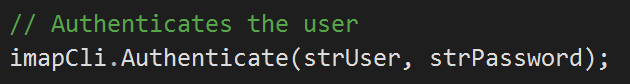
|  |  |
| --- | --- |
| Fonction | SynchronizeFiles() - Événement |
| Paramètres | **Object** source  **FileSystemEventArgs** e |
| Retour | - |
| Description détaillée | Cette fonction est appelée lorsque la surveillance du dossier racine rapporte une modification.  La mise à jour des listes de fichiers est lancée puis, une fois les liste à jour, c’est la comparaison de ces listes qui est exécutée. |

|  |  |
| --- | --- |
| Fonction | RefreshWindowElements() - Événement |
| Paramètres | **Object** sender  **EventArgs** e |
| Retour | - |
| Description détaillée | Timer qui vérifie que le dossier racine existe toujours et qui met constamment à jour le statut de l’ |

### Connexion au serveur IMAP et interactions boite mail

Pour que des communications avec la boite mail de l’utilisateur puissent se faire, il faut commencer par établir une connexion continue avec le serveur IMAP puis authentifier l’utilisateur. Toutes les interactions avec le serveur IMAP et la boite mail se font depuis la classe *MailManager*.

Connexion au serveur IMAP



Authentification utilisateur

Dans cette classe plusieurs fonctions sont disponibles pour interagir avec la boite mail. Il est possible d’établir une connexion, de se déconnecter, d’envoyer ou encore de recevoir des mails.

*MailManager* est une classe statique, c’est-à-dire qu’elle ne nécessite pas d’être instanciée pour être utilisée. Au lancement de l’application, c’est depuis la fenêtre de login qu’elle est appelée. Une connexion en IMAP au serveur mail est initiée afin que les transferts de mails puissent se faire.

Par la suite, les fonctions d’envoi et de réception de mails sont utilisées lors de la synchronisation entre le dossier racine local choisi par l’utilisateur et la boite mail.

Ci-dessous la liste des fonctions disponibles à l’utilisation dans la classe *MailManager*.

|  |  |
| --- | --- |
| Fonction | ConnectIMAP() |
| Paramètres | **String** strServer  **String** strPort  **String** strUser  **String** strPassword |
| Retour | **Bool** - Résultat de la connexion |
| Description détaillée | Cette fonction est appelée depuis la fenêtre de Login une fois que le formulaire de connexion a été validé.  C’est dans ce code que la connexion au serveur IMAP est initiée.  Des messages d’erreur et un booléen sont envoyés en retour en cas de problème de connexion au serveur IMAP et à la boite mail. |

|  |  |
| --- | --- |
| Fonction | DisconnectIMAP() |
| Paramètres | - |
| Retour | - |
| Description détaillée | Simple fonction de déconnexion du serveur IMAP. |

|  |  |
| --- | --- |
| Fonction | CreateStorageFolder() |
| Paramètres | - |
| Retour | - |
| Description détaillée | Ce code est appelé directement après que la connexion IMAP soit établie. Il consiste à vérifier que le dossier de mails « MailStorage » existe dans la boite mail de l’utilisateur. Dans le cas contraire, il est créé. |

|  |  |
| --- | --- |
| Fonction | SendMailToStorage() |
| Paramètres | **String** strSubject  **String** strBody |
| Retour | - |
| Description détaillée | C’est la fonction d’ajout de mail à la boite mail. Un nouveau mail est créé à partir des paramètres donnés puis il est placé dans le dossier « MailStorage » de la boite mail. |

|  |  |
| --- | --- |
| Fonction | UpdateIndexMail() |
| Paramètres | **String** indexBody |
| Retour | - |
| Description détaillée | Cette fonction va mettre à jour le mail d’index des dossiers.  Ce code est appelé à chaque synchronisation. |

|  |  |
| --- | --- |
| Fonction | DeleteMailInStorage() |
| Paramètres | **AppFile** fileToDelete |
| Retour | - |
| Description détaillée | Supprime un mail spécifique. La liste des mails présents dans la boite mail est parcourue jusqu’à trouver le mail et le supprimer. |

|  |  |
| --- | --- |
| Fonction | GetAllMailSubjects() |
| Paramètres | - |
| Retour | **List<String>** - Liste de tous les sujets |
| Description détaillée | Récupère le sujet de tous les mails présents dans le dossier MailStorage de la boite mail.  L’avantage de cette fonction est qu’elle ne parcourt que les sujets des mails, ce qui évite de télécharger le mail complet pour en retirer des informations. Le processus est donc particulièrement rapide. |

|  |  |
| --- | --- |
| Fonction | GetAllMails() |
| Paramètres | - |
| Retour | **List<MimeMessage>** - Liste avec tous les mails |
| Description détaillée | Cette fonction diffère légèrement de « GetAllMailSubjects() » car celle-ci va télécharger intégralement tous les mails présents dans le dossier MailStorage de la boite mail.  Une liste contenant tous les messages est retournée. |

|  |  |
| --- | --- |
| Fonction | GetOneMail() |
| Paramètres | Bool – isIndex = true  AppFile – fileToGet = null |
| Retour | **MimeMessage** – Le mail téléchargé |
| Description détaillée | Cette fonction permet de spécifier un mail à télécharger dans la boite mail.  Les paramètres données peuvent spécifier si le mail voulu est simplement le mail d’index des dossiers ou si c’est un mail de fichier qui est recherché.  Le mail téléchargé est ensuite retourné sous forme de « MimeMessage » |

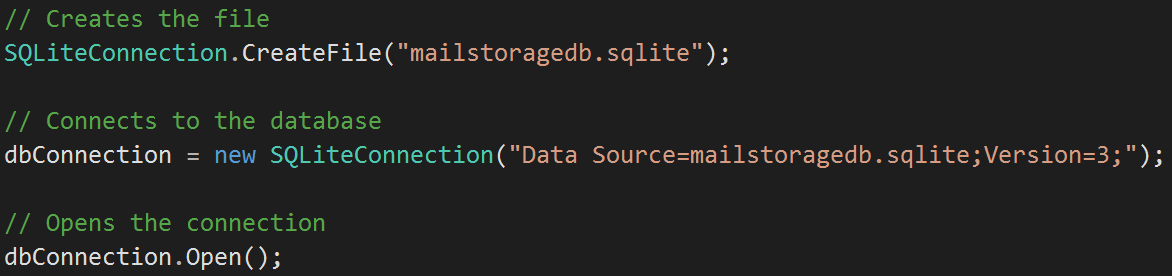
|  |  |
| --- | --- |
| Fonction | GetMailboxQuota() |
| Paramètres | - |
| Retour | **FolderQuota** – Informations sur la boite mail |
| Description détaillée | Récupère les informations de la boite mail et les renvois sous forme de « FolderQuota » |

### Interactions avec la base de données

L’utilisation d’une base de données dans le projet permet l’enregistrement des données de l’utilisateur afin que la connexion au serveur IMAP se fasse automatiquement au lancement de l’application.

La classe *DbManager* se charge d’effectuer toutes les interactions avec la base de données. Il est possible de créer la table de données, de retirer ou d’ajouter des données. L’API utilisée est SQLite.

Les seules utilisations de la base de données se font lors de l’étape de connexion au serveur IMAP, lors du login utilisateur. Les informations de l’utilisateur sont stockées dès que la connexion IMAP est établie, ce qui permet une reconnexion automatique au prochain lancement de l’application.

Lors du premier lancement de l’application, la base de données est créée puis une connexion est ouverte pour y insérer et récupérer des données.

La librairie SQLite permet des interactions SQL simples avec la base de données.

Ci-dessous, les fonctions disponibles dans la classe *DbManager*.

|  |  |
| --- | --- |
| Fonction | DbManager - Constructeur |
| Paramètres | - |
| Retour | - |
| Description détaillée | Dans ce constructeur, une connexion à la base de données SQLite est lancée.  Si la table AppData n’existe pas, elle est immédiatement créée afin que les données puissent y être stockées. |

|  |  |
| --- | --- |
| Fonction | UpdateUserData |
| Paramètres | **String** serverName  **String** serverPort  **String** userMail  **String** userPassword  **String** dirPath |
| Retour | - |
| Description détaillée | Cette fonction va insérer tous les paramètres entrants dans la table AppData de la base de données.  Les données stockées correspondent au nom et port du serveur IMAP, au mail et mot de passe de l’utilisateur et au dossier racine choisi. |

|  |  |
| --- | --- |
| Fonction | GetCurrentUserData() |
| Paramètres | - |
| Retour | **List<String>** - Données de l’utilisateur |
| Description détaillée | Permet de récupérer les données de l’utilisateur qui sont stockées dans la table AppData.  Cette fonction retourne une liste avec les valeurs nécessaires pour remplir entièrement et automatiquement le formulaire de connexion. |

|  |  |
| --- | --- |
| Fonction | ExecuteSQLQuery() |
| Paramètres | - |
| Retour | **String** – Requête SQL à exécuter |
| Description détaillée | Exécute une requête SQL dans la base de données.  La requête peut être un INSERT ou un DELETE mais cette fonction ne retourne aucune valeur. Il n’est donc pas possible d’effectuer un SELECT. |

### La synchronisation automatique des fichiers

La synchronisation des fichiers « dossier local – boite mail » est le cœur du programme. Afin que les fichiers présents dans le dossier racine choisi par l’utilisateur correspondent en continu aux fichiers présents dans la boite mail, deux listes de fichiers sont constamment tenues à jour afin de répertorier tous les fichiers présents aux deux endroits.

Liste de fichiers mail

Liste de fichiers locaux

Boite mail de l’utilisateur

Dossier racine choisi par l’utilisateur

Lorsque les deux listes sont à jour, elles sont comparées. Les fichiers présents dans la liste locale qui sont manquants dans la liste distante sont envoyés dans la boite mail tandis que les fichiers manquants dans la liste locale mais présents dans la liste distante sont supprimés de la boite mail.

Liste de fichiers locaux

Ficher manquant

Liste de fichiers locaux

Ficher présent

Ajout du fichier à la boite mail

Fichier manquant

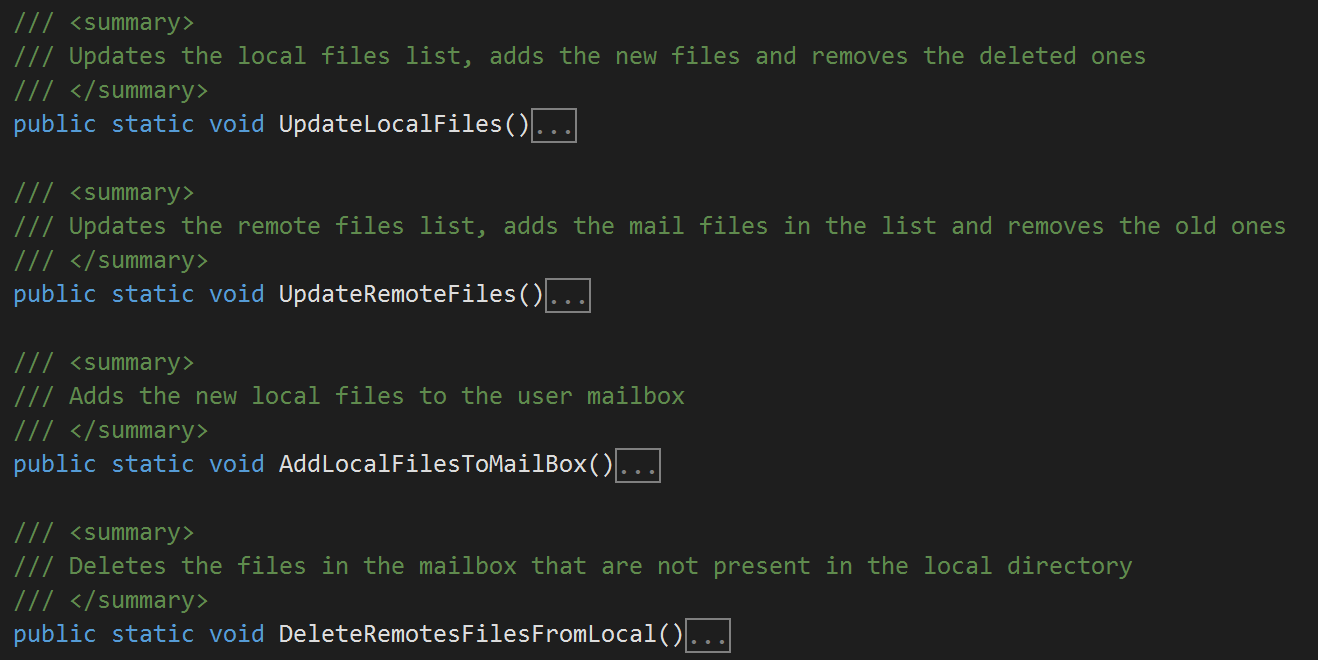
Liste de fichiers mail

Suppression du fichier de la boite mail

Fichier présent

Liste de fichiers mail

L’application a été pensée pour être utilisée par une seule personne à la fois. Cependant, une fonctionnalité supplémentaire permettant de gérer une utilisation simultanée du programme est envisageable.

Toutes ces interactions de fichiers sont gérées par quatre fonctions principales au sein de la classe *FilesManager*.

Suppressions dans la boite mail

Mise à jour de la liste distante

Mise à jour de la liste locale

Ajout des fichiers à la boite mail

La classe *FilesManager* est une des classes les plus importantes du programme. C’est dans cette classe que sont gérées toutes les interactions avec les fichiers, que ce soit au niveau de l’indexation ou au niveau de la transformation des fichiers en texte.

Ci-dessous la liste de toutes les fonctions de la classe *FilesManager* et leur description.

|  |  |
| --- | --- |
| Fonction | InitialSynchronisation() |
| Paramètres | - |
| Retour | - |
| Description détaillée | La fonction de synchronisation initiale perme le téléchargement des fichiers de la boite mail lorsque le dossier racine est spécifié r l’utilisateur pour la première fois.  Cette fonction va récupérer chaque fichier de la boite mail et le placer dans le dossier local sous forme de fichier.  Le fichier d’indexation des dossiers va aussi être téléchargé afin de recréer l’arborescence des dossiers dans le nouveau dossier local. |

|  |  |
| --- | --- |
| Fonction | StartUpdateFromMailBox() |
| Paramètres | - |
| Retour | - |
| Description détaillée | Cette fonction représente le processus de synchronisation au démarrage.  À chaque lancement de l’application, cette fonction est appelée et va vérifier l’état de la boite mail.  Si de nouveaux fichiers ou des plus récents s’y trouvent, ils sont téléchargés et placés dans le dossier local. Les fichiers présents dans le dossier local mais absents de la boite mail sont supprimés du dossier.  En comptant la fonction de synchronisation initiale, c’est l’un des seuls moments ou la priorité est donnée à la boite mail pour la création et suppression de fichiers. |

|  |  |
| --- | --- |
| Fonction | UpdateLocalFiles() |
| Paramètres | - |
| Retour | - |
| Description détaillée | Cette fonction représente l’étape de mise à jour de la liste des fichiers locaux.  Pour commencer, les fichiers présents dans le dossier local sont tous listés.  Ensuite, ceux qui ont été supprimés et qui sont encore présents dans la liste sont retirés.  Et pour finir, nouveaux fichiers listés sont ajoutés à la liste de fichiers locaux. |

|  |  |
| --- | --- |
| Fonction | UpdateRemoteFiles() |
| Paramètres | - |
| Retour | - |
| Description détaillée | C’est dans cette fonction que les fichiers de la boite mail sont listés dans la liste distante.  Tout d’abord, tous les sujets des mails sont récupérés, puis chaque information est séparée.  Ensuite, une instance de fichier « AppFile » est créée avec chaque donnée récupérée.  Et pour finir, ces nouvelles instances sont placées dans la liste distante. |

|  |  |
| --- | --- |
| Fonction | AddLocalFilesToMailBox() |
| Paramètres | - |
| Retour | - |
| Description détaillée | Fonction faisant partie de l’étape de comparaison des listes. Cette fonction va vérifier quels fichiers sont présents dans la liste locale mais absents de la liste distante.  Ces fichiers vont être transformés en texte Base64 puis ajoutés à la boite mail sous forme de mail. |

|  |  |
| --- | --- |
| Fonction | DeleteRemoteFilesFromLocal() |
| Paramètres | - |
| Retour | - |
| Description détaillée | Deuxième fonction faisant partie de l’étape de comparaison des listes, c’est dans ce code que les fichiers qui ne sont plus présents dans le dossier local sont supprimés de la boite mail.  Les deux listes sont comparées et tous les fichiers présents uniquement dans la liste distante sont supprimés de la boite mail afin d’égaliser les deux listes. |

|  |  |
| --- | --- |
| Fonction | UpdateDirectoriesIndex() |
| Paramètres | - |
| Retour | - |
| Description détaillée | Etablit l’arborescence complète des dossiers dans le dossier racine sous forme de texte et met à jour le mail d’indexation des dossiers. |

|  |  |
| --- | --- |
| Fonction | ConvertFileTo64() |
| Paramètres | **AppFile** fileToConvert |
| Retour | **String** – Le fichier en texte Base64 |
| Description détaillée | Convertit le fichier donné en paramètre en chaine de caractères Base64. Le texte est retourné à la fin de l’exécution de la fonction. |

|  |  |
| --- | --- |
| Fonction | AddLocalFileToList() |
| Paramètres | **String** strFilePath |
| Retour | - |
| Description détaillée | Cette fonction récupère les informations d’un fichier dont le chemin d’accès est donné en paramètres pour ensuite créer une nouvelle instance de « AppFile » qui sera par la suite ajoutée à la liste des fichiers du dossier local. |

#### Mise à jour de la liste locale

La liste locale contient tous les fichiers qui sont présents dans le dossier racine choisi par l’utilisateur. À chaque synchronisation, la liste va se mettre à jour en éliminant les fichiers qui ne sont plus présents dans le dossier local et en y ajoutant les nouveaux.

Dossier racine choisi par l’utilisateur

Liste de fichiers locaux

La liste des fichiers du dossier local est récupérée grâce à la classe *Directory*, qui permet de chercher des fichiers récursivement à partir d’un emplacement donné. Les informations des fichiers sont ensuite retournées sous forme de *FileInfo* dans un tableau.

Le tableau de fichiers est ensuite parcouru. Dans le cas où un fichier est présent dans la liste locale mais pas dans le tableau de fichiers (donc pas dans le dossier local), ce fichier est supprimé de la liste locale.

Si au contraire un fichier est présent dans le dossier local mais pas dans la liste locale, il est ajouté à cette dernière.

|  |  |  |
| --- | --- | --- |
| Le fichier | Est dans le dossier | N’est pas dans le dossier |
| **Est dans la liste** | Ne rien faire | Supprimer de la liste |
| **N’est pas dans la liste** | Ajouter à la liste | Ne rien faire |

Une fois la comparaison entre la liste locale et le dossier racine effectuée, la liste locale se retrouve mise à jour avec une liste complète et correcte des fichiers présents dans le dossier racine.

#### Mise à jour de la liste distante

La mise à jour de la liste distante consiste à vérifier les fichiers présents sur la boite mail de l’utilisateur et changer les éléments de la liste afin qu’ils correspondent exactement à l’état de la boite mail.

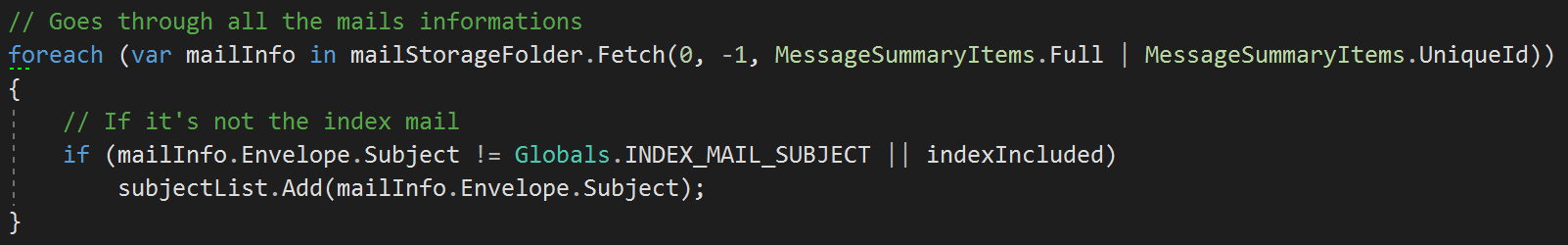
Liste de fichiers mail

Boite mail de l’utilisateur

Le processus de mise à jour de la liste distante ressemble à celui de la liste locale. Cependant, pour pouvoir mettre la liste distante à jour, il faut pouvoir vérifier quels fichiers sont sur la boite mail et cela implique des interactions avec le serveur mail et donc la librairie *MailKit* et la classe *MailManager*.

Pour récupérer l’état des fichiers dans la boite mail, le programme utilise la fonction *GetAllMailSubjects()* qui va récupérer tous les sujets des mails dans la boite mail et le renvoyer en output.

Parcourir chaque mail

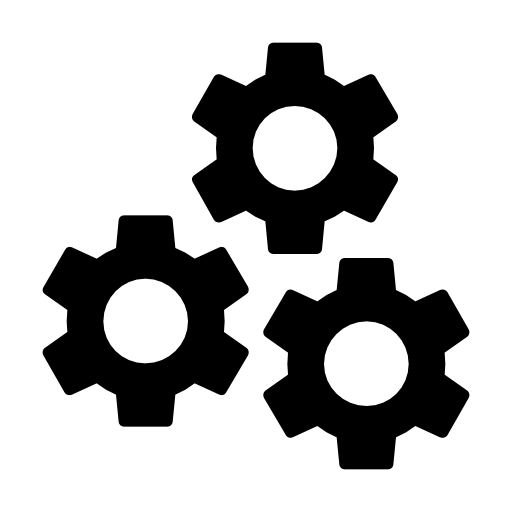


Récupération du sujet et ajout du sujet dans l’output

Une fois les sujets récupérés, les noms des fichiers sont isolés et la comparaison de l’état de la boite mail et de la liste distante peut se faire. Comme pour la liste locale, les fichiers présents dans la boite mail mais pas dans la liste distante sont ajoutés à la liste tandis que les fichiers présents dans la liste distante mais pas dans la boite mail sont supprimés de la liste.

#### Envoi des nouveaux fichiers sur la boite mail

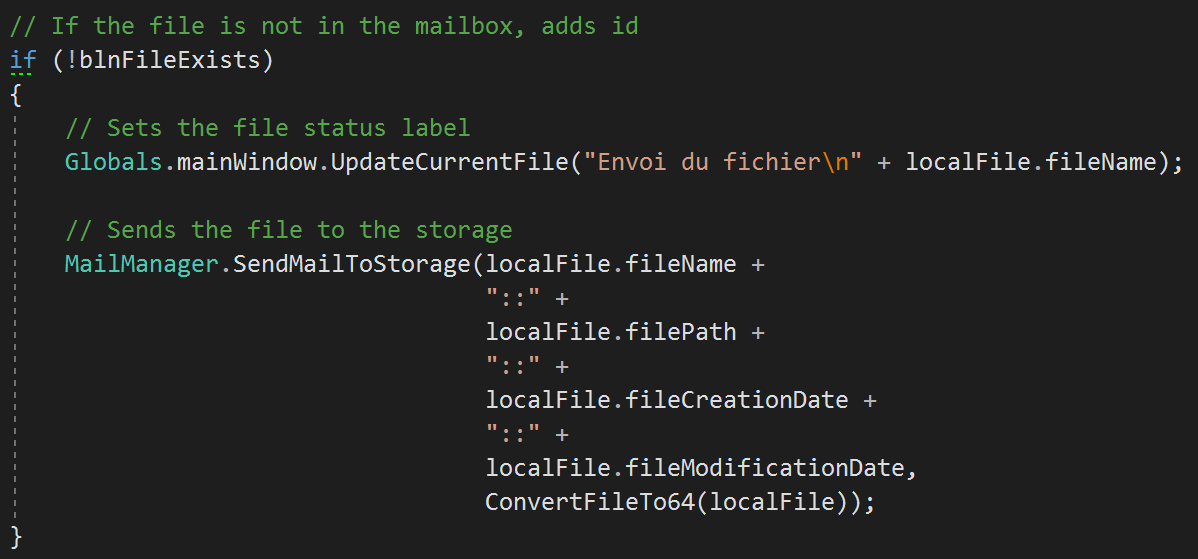
L’envoi de fichiers sur la boite mail de l’utilisateur se fait lors de l’étape de comparaison de la liste locale avec la liste distante.

Lorsque les deux listes sont à jour, la fonction *AddLocalFilesToMailBox()* va parcourir chaque élément de la liste locale et vérifier que cet élément possède un équivalent dans la liste distante. Si ce n’est pas le cas, le fichier en question est transformé en texte Base64 puis envoyé sou forme de mail dans la boite mail de l’utilisateur.

Transformation du fichier en Base64 et envoi du mail dans la boite mail.

Ne rien faire, le fichier est déjà dans la boite mail.

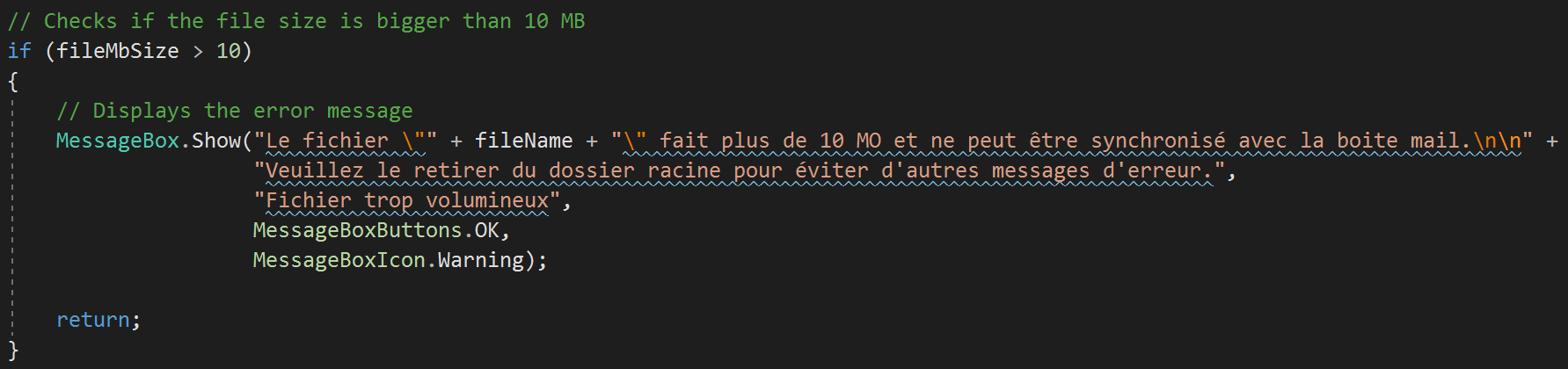
Le fichier est présent dans les deux listes ?

Les fichiers manquants dans la boite mail y sont ainsi ajoutés et à la prochaine synchronisation, les deux listes seront équivalentes.

Condition, si le fichier n’est pas dans la boite mail

Envoi du mail de fichier

Il existe cependant une contrainte pour l’envoi de ces fichiers : la taille. En effet, la taille du fichier à envoyer ne doit pas dépasser 10 Mo. Cette restriction est due à des raisons de temps d’envoi et de limitation de taille de mail dans la boite mail.

Les fichiers de plus de 10 Mo sont simplement détectés lors de la mise à jour de la liste locale et n’y sont pas ajoutés.

Retour de la fonction sans ajout du fichier à la liste

Affichage du message d’erreur à l’écran

Condition, si le fichier fait plus de 10 Mo

Une fonctionnalité supplémentaire a été envisagée lors de l’analyse pour permettre l’envoi de fichiers sans limitation de taille. Cependant, le manque de temps pour le projet n’a pas permis la réalisation de cette fonctionnalité. Pour plus d’informations, voir la rubrique « Fonctionnalités supplémentaires à implémenter » dans la conclusion du rapport.

#### Suppression des fichiers de la boite mail

Deuxième étape de la comparaison des listes, la suppression de mails de fichiers de la boite mail se fait lorsqu’un fichier a été supprimé, renommé ou édité.

Chaque élément de la liste distante et parcouru et pour chacun de ces éléments, le programme vérifie qu’il existe un équivalent dans la liste locale. Si ce n’est pas le cas, le fichier de la liste distante est identifié dans la boite mail puis supprimé.

Suppression du fichier mail

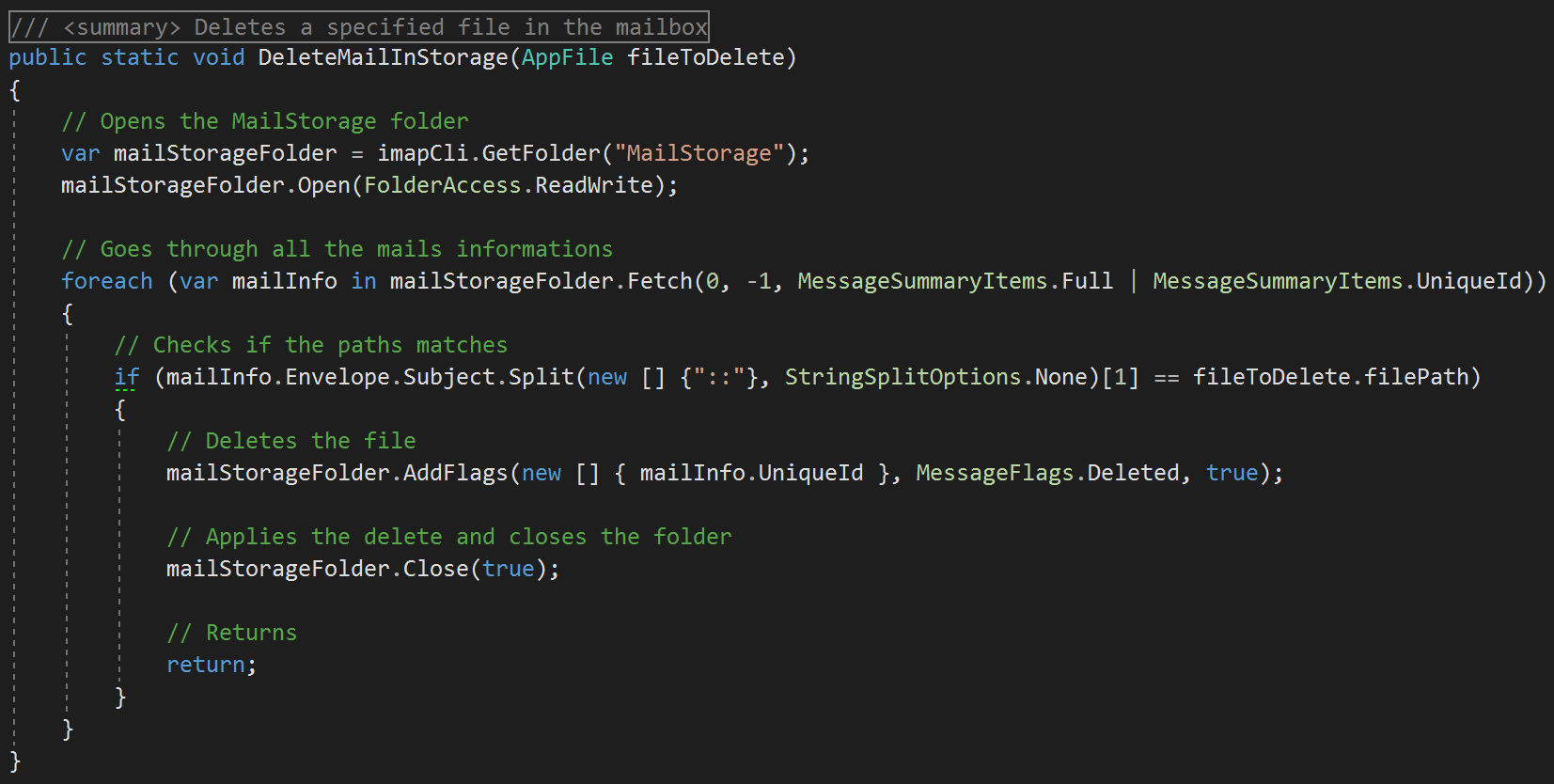
Condition, si le fichier n’est pas dans la liste locale

La suppression du mail de fichier se fait depuis la classe *MailManager* car cela implique une interaction avec la boite mail.

Lorsqu’un fichier est modifié dans le dossier local, la date de modification est prise en compte dans la comparaison des listes ce qui fait que le fichier est plus récent et doit être mis à jour dans la boite mail. C’est à ce moment que la suppression de fichier intervient. L’ancienne version du fichier est supprimée de la boite mail afin que la nouvelle version puisse se créer.

Dans le cas d’un renommage de fichier, le principe reste le même. Le fichier ne possède plus le même nom et le programme l’identifie comme un nouveau fichier. Le fichier avec l’ancien nom est donc supprimé de a boite mail et la nouvelle version avec le nouveau nom y est créée.

La suppression d’un fichier spécifié se fait dans la classe *MailManager*. Tout d’abord le programme parcourt tous les sujets des mails de la boite mail et compare les noms de fichiers avec le fichier recherché.

Une fois que le fichier est trouvé, l’uuid du mail correspondant est récupéré et le tag « supprimé » est appliqué au mail. Les changements de la boite mail sont ensuite enregistrés. Le mail du fichier spécifié est correctement supprimé de la boite mail de l’utilisateur.

Lecture de tous les sujets de mail

Fermeture du dossier « MailStorage » et application des changements

Application du tag « Supprimé » au mail spécifié

Ouverture du dossier « MailStorage »

### Synchronisation initiale

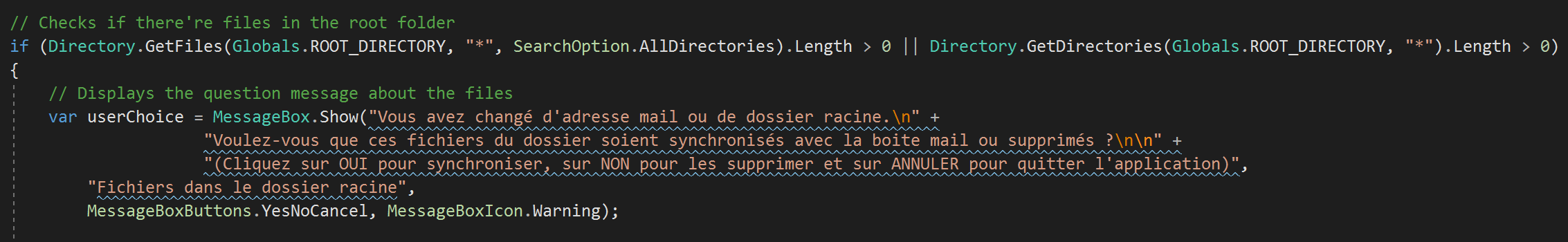
La synchronisation initiale intervient lorsque l’utilisateur lance le programme pour la première fois, ou plus précisément, lorsqu’il définit un nouveau dossier racine au programme.

Le but de cette synchronisation est de s’informer sur l’état de l’arborescence et des fichiers dans la boite mail pour ensuite tous les télécharger dans le nouveau dossier local.

Cette synchronisation permet de remplir le dossier racine avec les fichiers qui ne s’y trouvent pas et qui pourtant ont été ajoutés par le passé à la boite mail.

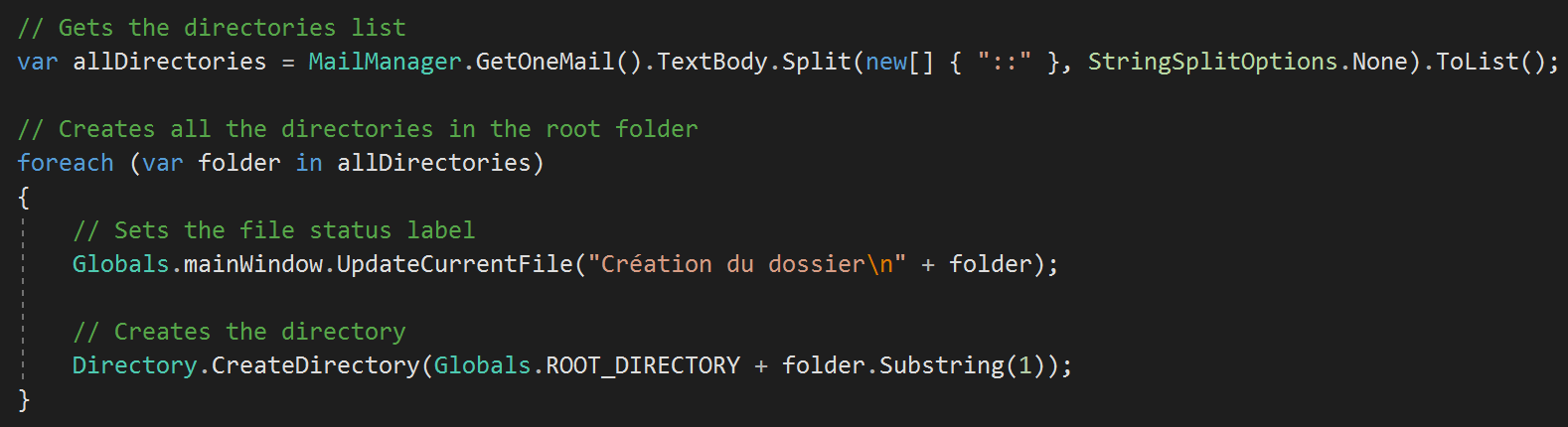
Un exemple est l’utilisation du programme sur deux ordinateurs différents. Le premier se charge de placer de nouveaux fichiers dans la boite mail. Lorsque l’utilisateur voudra utiliser le programme sur le deuxième ordinateur, il va spécifier un autre dossier racine. La synchronisation initiale va vérifier les fichiers présents dans la boite mail et va tous les télécharger dans le nouveau dossier local.

La fonction s’exécute en passant par plusieurs étapes.

Premièrement, l’application vérifie sir des fichiers sont déjà présents dans le dossier racine et demande à l’utilisateur s’il veut les supprimer ou les synchroniser.

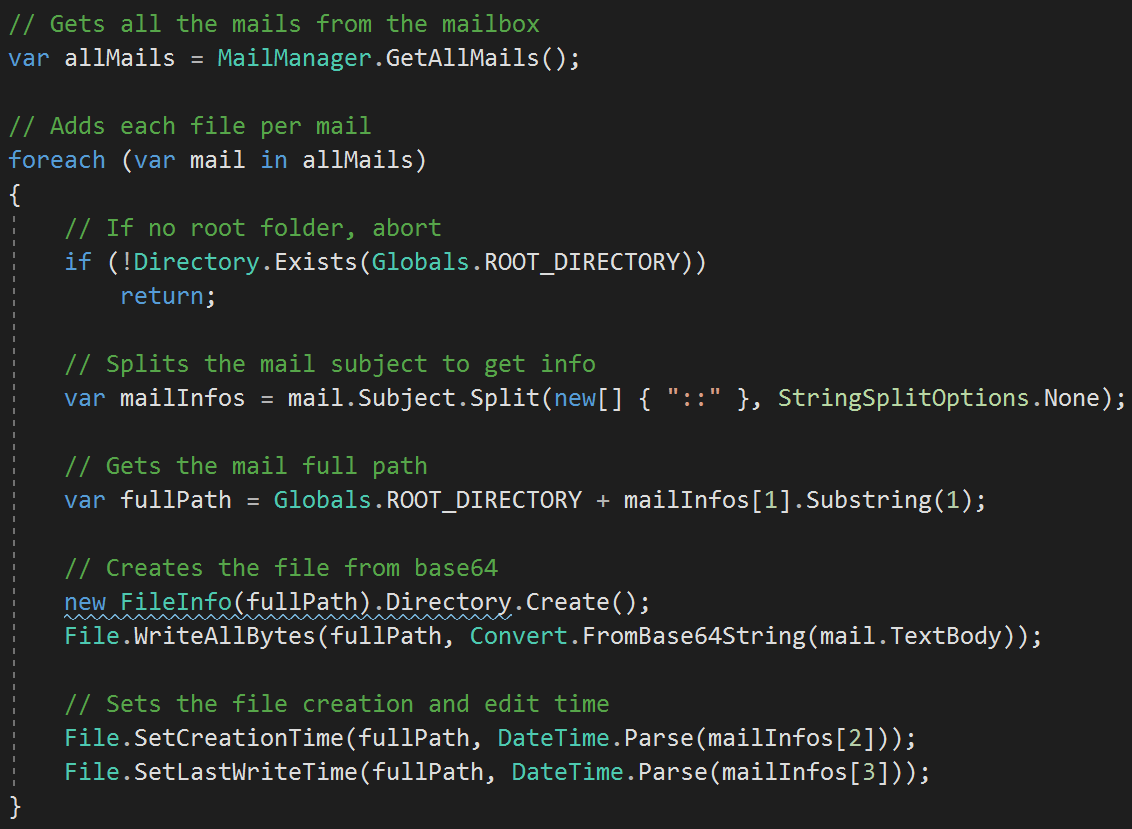
Vérifie s’il y a des fichiers dans le dossier local

Message d’avertissement

Ensuite, le mail d’arborescence des dossiers est récupéré et les dossiers sont créés dans le dossier racine.

Création des dossiers

Récupération de l’arborescence

Et pour finir, tous les fichiers mail sont téléchargés et placés dans le dossier local.

Crée le nouveau fichier

Transforme le texte Base64 en fichier

Parcours tous les mails

Récupère tous les mails

Les mails récupérés sont sous forme de texte Base64. La fonction *Convert.FromBase64String()* est utilisée pour récupérer le binaire qui est utilisé pour créer les fichiers dans le dossier local.

Les dates de création et de modification sont ensuite appliquées au nouveau fichier crée afin qui soit identique au fichier présent dans la boite mail.

Les informations du fichier sont stockées dans le sujet su mail et séparée par les caractères « :: ». Une fois le sujet du mail séparé en plusieurs éléments et placé dans un tableau, cela permet d’accéder aux informations du fichier pour le recréer, lui donner le nom correct et les dates de création et de modification égales à celles de la boite mail.

### Synchronisation au démarrage

Cette synchronisation est automatiquement lancée à chaque lancement du programme et consiste à donner la priorité à la boite mail.

Une synchronisation normale supprime les fichiers dans la boite mail et les ajoute en fonction de ce que le dossier racine contient. Dans la synchronisation au démarrage, c’est en fonction de ce que la boite mail contient que les fichiers sont ajoutés ou supprimés au dossier local. Les rôles sont inversés.

Une telle synchronisation permet de mettre à jour l’état du dossier local dans le cas où le programme est utilisé à plusieurs endroits en même temps.

Par exemple si un fichier est rajouté à la boite mail sur l’ordinateur A et que l’utilisateur lance le programme sur l’ordinateur B, la synchronisation au démarrage va se lancer et le nouveau fichier va être téléchargé dans le dossier racine de l’ordinateur B. Sans cette synchronisation, les fichiers ne seraient jamais téléchargés ce qui rendrait l’utilisation du programme impossible.

Ajout de fichier

Téléchargement du fichier

Boite mail

Ordinateur B

Ordinateur A

La synchronisation au démarrage est similaire à la synchronisation initiale mais contrairement à cette dernière, elle va aussi supprimer les fichiers dans le dossier local qui ne sont pas dans la boite mail.

Cette synchronisation initiale se fait au tout début de l’exécution du programme et ne dure généralement pas très longtemps. Aucun envoi de fichier n’y est fait. La fonction de synchronisation au démarrage est appelée *StartUpdateFromMailBox()*.

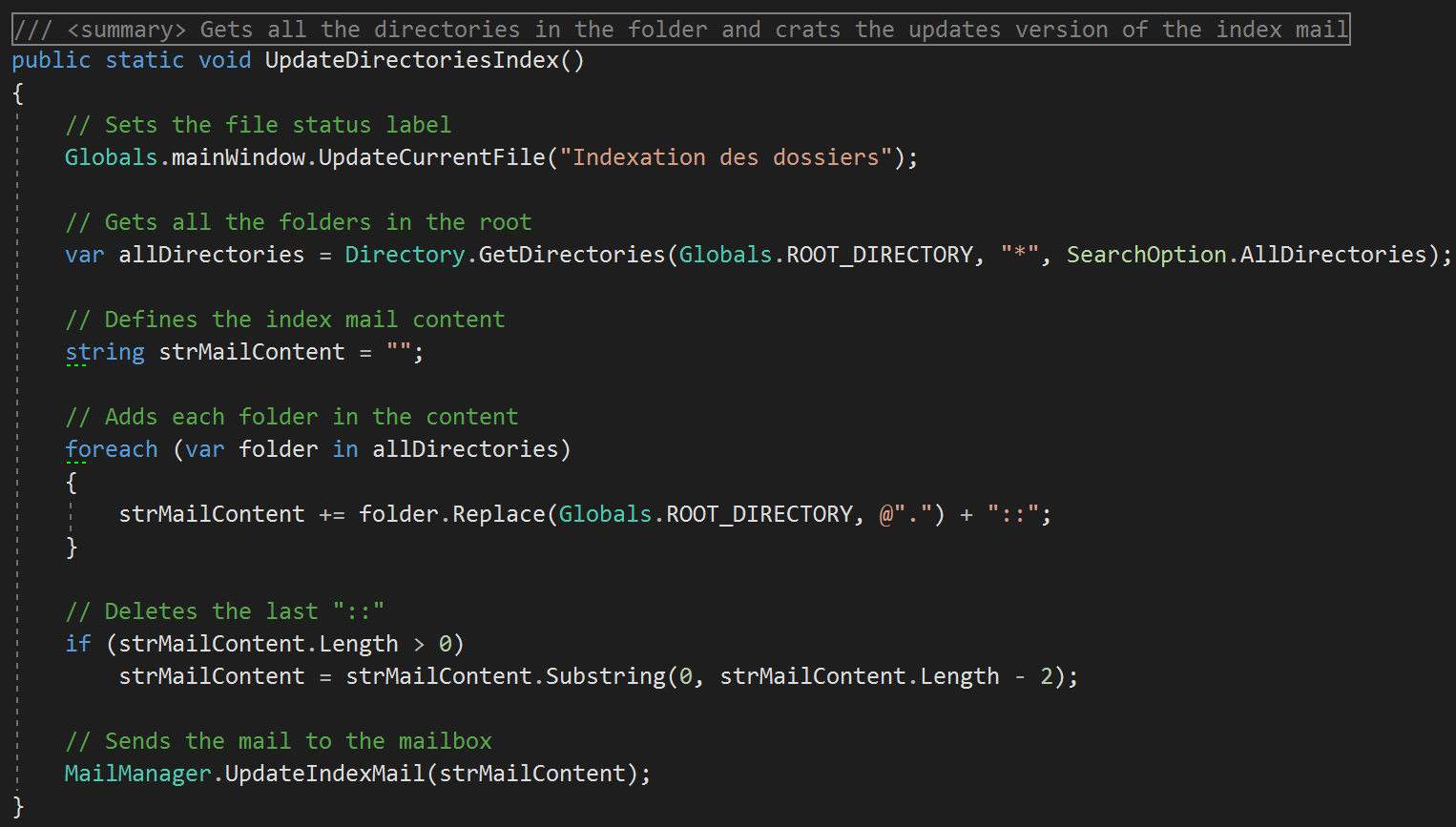
### Mail d’index des dossiers

Dans le dossier « MailStorage » de la boite mail, il existe un mail différent et qui n’a rien à voir avec les autres mails de fichiers : c’est le mail d’arborescence des dossiers.

Ce mail contient simplement la liste complète de tous les dossiers présents dans le dossier racine. L’utilité d’un tel mail est de pouvoir recréer l’arborescence lorsqu’un nouveau dossier racine est choisi.

Ce mail d’arborescence est mis à jour à chaque synchronisation et est appelé à chaque synchronisation initiale.

Sa structure est très simple, il possède le chemin relatif de chaque dossier présent dans le dossier racine. Chaque chemin est séparé par les caractères « :: ».

La fonction qui se charge de mettre à jour le mail d’index se trouve dans la classe *FilesManager* et se nomme *UpdateDirectoriesIndex()*. Dans ce code, l’ancien mail est supprimé et remplacé par le nouveau.

Remplace l’ancien mail par le nouveau

Ajoute le chemin de chaque dossier au nouveau mail d’index

Récupère tous les dossiers à la racine

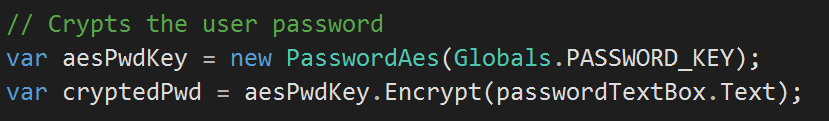
### Cryptage du mot de passe de l’utilisateur

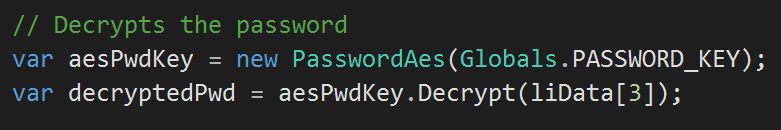
La première question qui vient à l’esprit est « pourquoi avoir choisi le cryptage au lieu du hachage pour un mot de passe ».

La réponse à cette question est plutôt simple. Le mot de passe de l’utilisateur nécessite d’être protégé, il ne peut donc pas être stocké en clair dans la base de données. Cependant, la fonctionnalité de connexion automatique qui permet au programme de se connecter automatiquement au compte mail de l’utilisateur nécessite que le programme entre lui-même le mot de passe dans le champ de connexion au serveur IMAP.

Le problème du hachage dans cette démarche est qu’il est impossible de récupérer le mot de passe automatiquement lorsqu’il a été haché. Pour pouvoir le récupérer, il faut que l’utilisateur l’écrive dans le formulaire de connexion, ce qui casse la fonctionnalité de connexion automatique.

La deuxième solution est alors le cryptage. La solution choisie est un cryptage AES du mot de passe de l’utilisateur avec une clé de cryptage stockée en local dans le programme. La librairie utilisée se nomme *csharp-aes*.

Cette API permet le cryptage d’une chaine de caractères très facilement.

Le cryptage du mot de passe se fait en récupérant la clé de cryptage, puis en encryptant le mot de passe. L’output consiste en une autre chaine de caractères qui représentent le mot de passe crypté. Cette chaine est placée dans la base de données.

Le décryptage est très similaire au cryptage, la clé de cryptage est récupérée puis la chaine de caractères extraite de la base de données est décryptée afin que le mot de passe puisse être récupéré en sortie.

## Dossier des tests

### Scénario N°1 : Connexion

|  |  |
| --- | --- |
| Étape | Description |
| Entrée | Fenêtre de connexion à l’application |
| Quand | Saisie du compte mail, do mot de passe et du dossier racine |
| Résultat | Connexion au compte mail et affichage de la fenêtre de traitement des fichiers |

|  |  |
| --- | --- |
| Résultat | OK |
| Remarque | |
| La connexion fonctionne avec différents serveurs IMAP, par contre les données de l’espace disponible ne sont pas accessibles pour tous les serveurs.  Cependant, certains serveurs mail sont moins stables que d’autres. L’application semble fonctionner mieux avec le serveur IMAP de Gmail tandis que d’autres serveurs comme Outlook peuvent parfois terminer la connexion sans aucune raison apparente. | |

### Scénario N°2 : Synchronisation

|  |  |
| --- | --- |
| Étape | Description |
| Entrée | Bouton de synchronisation manuelle des fichiers |
| Quand | Clic sur le bouton de synchronisation |
| Résultat | Mise à jour et synchronisation des fichiers « dossier - boite mail » |

|  |  |
| --- | --- |
| Résultat | OK |
| Remarque | |
| Lance la synchronisation manuelle. Les fichiers du dossier local et de la boite mail sont synchronisés. | |

### Scénario N°3 : Changement de compte

|  |  |
| --- | --- |
| Étape | Description |
| Entrée | Fenêtre de connexion à l’application |
| Quand | Changement du compte mail, mot de passe et dossier racine |
| Résultat | Connexion au compte mail et affichage de la fenêtre de traitement des fichiers |

|  |  |
| --- | --- |
| Résultat | OK |
| Remarque | |
| La connexion au serveur mail est interrompue, la fenêtre de login est affichée et une fois la reconnexion effectuée, si l’utilisateur a décidé de changer de dossier racine, une synchronisation initiale est lancée. | |

### Scénario N°4 : Espace disponible

|  |  |
| --- | --- |
| Étape | Description |
| Entrée | Fenêtre de traitement des fichiers |
| Quand | À tout moment |
| Résultat | L’espace disponible affiché correspond à l’espace disponible dans la boite mail |

|  |  |
| --- | --- |
| Résultat | Fonctionnel mais limité |
| Remarque | |
| L’espace disponible s’affiche lorsque l’application est connectée au serveur IMAP.  Cependant, certains serveurs IMAP (Outlook par exemple) n’offrent pas la possibilité de récupérer l’espace disponible ce qui rend impossible l’affichage de cette donnée dans le programme. | |

### Scénario N°5 : Synchronisation initiale

|  |  |
| --- | --- |
| Étape | Description |
| Entrée | Dossier local |
| Quand | Après connexion à la boite mail |
| Résultat | Création automatique des fichiers depuis la boite mail |

|  |  |
| --- | --- |
| Résultat | OK |
| Remarque | |
| Lors de la première connexion ou lorsque le dossier racine est changé, la synchronisation initiale est lancée et va récupérer tous les fichiers et les dossiers stockés dans la boite mail pour les ajouter en local. | |

### Scénario N°6 : Création de fichiers

|  |  |
| --- | --- |
| Étape | Description |
| Entrée | Dossier local |
| Quand | Ajout d’un fichier dans le dossier |
| Résultat | Ajout automatique du fichier dans la boite mail |

|  |  |
| --- | --- |
| Résultat | OK |
| Remarque | |
| Lorsqu’un fichier est créé dans le dossier local, le programme détecte automatiquement le changement et lance une synchronisation automatique.  Le fichier est transformé en texte Base64 et est ensuite visible dans la boite mail dans le dossier MailStorage. | |

### Scénario N°7 : Suppression de mail

|  |  |
| --- | --- |
| Étape | Description |
| Entrée | Boite mail |
| Quand | Suppression d’un mail de fichier |
| Résultat | Recréation automatique du mail dans la boite mail |

|  |  |
| --- | --- |
| Résultat | OK |
| Remarque | |
| Dans le cas où un fichier est supprimé de la boite mail manuellement, il sera automatiquement recréé lors de la prochaine synchronisation. | |

### Scénario N°8 : Suppression de fichier

|  |  |
| --- | --- |
| Étape | Description |
| Entrée | Dossier local |
| Quand | Suppression d’un fichier dans le dossier local |
| Résultat | Suppression automatique dans la boite mail |

|  |  |
| --- | --- |
| Résultat | OK |
| Remarque | |
| De la même façon que pour la création d’un fichier, lorsqu’un fichier est supprimé, le changement est détecté automatiquement et une synchronisation est lancée.  Quelques instants après, le fichier apparait dans le dossier MailStorage de la boite mail. | |

### Scénario N°9 : Édition de fichier

|  |  |
| --- | --- |
| Étape | Description |
| Entrée | Dossier local |
| Quand | Édition d’un fichier dans le dossier local |
| Résultat | Mise à jour automatique du fichier dans la boite mail |

|  |  |
| --- | --- |
| Résultat | OK |
| Remarque | |
| Lorsqu’un fichier est modifié, le changement est détecté automatiquement par le programme et une synchronisation est lancée.  L’ancien fichier est d’abord supprimé de la boite mail, puis la nouvelle version est ajoutée. | |

### Scénario N°10 : Gestion des erreurs

|  |  |
| --- | --- |
| Étape | Description |
| Entrée | Fenêtre de connexion à l’application |
| Quand | Insertion d’une fausse adresse mail et mauvais mot de passe |
| Résultat | Affichage des messages d’erreur |

|  |  |
| --- | --- |
| Résultat | OK |
| Remarque | |
| Des messages d’avertissement s’affichent lorsque :   * Le serveur IMAP interdit la connexion * Lorsque l’authentification échoue * Le dossier MailStorage n’est pas accessible * Si des fichiers sont présents localement lors d’une synchronisation initiale * Les fichiers sont trop volumineux pour une synchronisation (10 Mo) * Si le dossier racine est supprimé * Si la connexion au serveur IMAP est perdue * Si l’utilisateur quitte l’application en cours de synchronisation * Si l’utilisateur retourne au login en cours de synchronisation * Si un fichier est impossible à supprimer * Message de synchronisation avant la fermeture de l’application | |

### Autres tests

|  |  |
| --- | --- |
| Description | Résultat |
| Vérification des champs de formulaire de connexion | **OK** |
| Explorateur pour sélectionner un dossier racine | **OK** |
| Le statut change lors d’une synchronisation | **OK** |
| La barre de chargement correspond à la valeur d’espace disponible | **OK** |
| Les dossiers sont recréés lors d’une synchronisation initiale | **OK** |
| Les fenêtres peuvent être déplacées et fermées | **OK** |
| La fenêtre principale peut être réduite et tourner en arrière-plan | **OK** |
| Le mot de passe de l’utilisateur est crypté | **OK** |
| Les fichiers sont lus en lecture seule pour éviter des conflits | **OK** |
| La synchronisation au démarrage télécharge les fichiers plus récents | **OK** |
| Les threads actifs n’entrent pas en conflit | **OK** |
| Les fichiers plus anciens sont supprimés au profit des plus récents | **OK** |
| Il n’y a pas de doublons de fichiers dans la boite mail | **OK** |
| L’état de traitement des fichiers en cours est affiché à l’écran | **OK** |

# 

Conclusion

# Conclusion

## Bilan des fonctionnalités demandées

Ci-dessous se trouvent toutes les fonctionnalités demandées dans le cahier des charges. L’état de la fonctionnalité indique si celle-ci a été implémentée ou abandonnée.

|  |  |
| --- | --- |
| Description de la fonctionnalité selon le cahier des charges | Etat |
| Configuration du serveur IMAP et du port | **OK** |
| Configuration du login et mot de passe utilisateur | **OK** |
| Affichage de l’espace disponible | **OK** |
| Créer un fichier | **OK** |
| Renommer un fichier | **OK** |
| Supprimer un fichier | **OK** |
| Ouvrir et éditer un fichier | **OK** |
| Conservation des métadonnées des fichiers | **OK** |
| Utilisation de l’explorateur du système d’exploitation | **OK** |
| Synchronisation des fichiers entre la boite mail et le dossier local | **OK** |
| Utilisation de l’espace de stockage de la boite mail | **OK** |
| Utilisation du protocole IMAP pour interagir avec la boite mail | **OK** |
| Fonctionnement sans gestion des droits utilisateur | **OK** |

L’intégralité des fonctionnalités demandées dans le cahier des charges ont été implémentées et testées.

D’autres fonctionnalités supplémentaires étaient prévues, cependant, le manque de temps disponible n’a pas permis leur implémentation. Pour plus d’informations sur ces fonctionnalités supplémentaires, voir la rubrique suivante.

## Fonctionnalités supplémentaires à implémenter

### Gestion des fichiers de grande taille

L’une des contraintes de l’utilisation de l’espace de stockage de la boite mail de l’utilisateur est la limitation de la taille d’un mail. Le problème qui survient avec l’application est que les fichiers envoyés ne peuvent dépasser 10 Mo au risque de recevoir un message de refus de la part de la boite mail.

Ce problème reste pour l’instant géré par un message d’avertissement indiquant que tout fichier dépassant la limite autorisée de 10 Mo ne sera pas synchronisé.

Cette erreur est causée par le fait d’utiliser un système basé sur la solution « 1 mail = 1 fichier ». Le programme n’attribue qu’un seul mail par fichier ce qui le rend parfois trop volumineux.

La solution pour résoudre ce souci de taille consiste à attribuer plusieurs mails lorsque le fichier est trop volumineux.

**De :** [mailstorage@mail.com](mailto:mailstorage@mail.com)

**À :** [moi@mail.com](mailto:moi@mail.com)

**Sujet :** C:\Users\name\file.txt

**Contenu :** 4HdjwijcUhI2f8dhJs

Fichier volumineux

**De :** [mailstorage@mail.com](mailto:mailstorage@mail.com)

**À :** [moi@mail.com](mailto:moi@mail.com)

**Sujet :** C:\Users\name\file.txt

**Contenu :** 0dw8Isosjfoj3wOsw

**De :** [mailstorage@mail.com](mailto:mailstorage@mail.com)

**À :** [moi@mail.com](mailto:moi@mail.com)

**Sujet :** C:\Users\name\file.txt

**Contenu :** R93eoSjdiw3919843

Chaque mail du même fichier possède un identifiant et lorsque le fichier doit être téléchargé, tous les mails correspondant à ce fichier sont téléchargés puis leur contenu en Base64 est mis bout à bout afin d’obtenir le fichier complet.

### Exécution simultanée sur plusieurs ordinateurs

Le programme a été conçu pour ne fonctionner qu’avec une seule instance à la fois. C’est-à-dire que la personne qui veut utiliser le programme sur deux ordinateurs de doit pas le lancer deux fois en même temps, au risque d’avoir des conflits de traitement de fichiers.

Une solution pour résoudre ce problème et gérer automatiquement l’utilisation simultanée du programme serait de donner à chaque fichier dans la boite mail une trace de quel ordinateur l’a créé, l’adresse mac par exemple.

Cela permettrait au programme en cours d’exécution de savoir quel fichier télécharger et quel fichier supprimer.

Sans cette trace de création, le programme verrait un fichier dans la boite mail qu’il ne possède pas (ayant été rajouté par un autre ordinateur) et le supprimerait.

Envoi des nouveaux fichiers du dossier local à la boite mail

Téléchargement des fichiers de la boite mail ayant été créés par n’importe quel autre ordinateur

Ordinateur A

Boite mail

Téléchargement des fichiers de la boite mail ayant été créés par n’importe quel autre ordinateur

Envoi des nouveaux fichiers du dossier local à la boite mail

Ordinateur B

## Bilan personnel

La conception de ce projet s’est faite en plusieurs étapes dans lesquelles j’ai dû organiser, planifier et gérer les tâches à accomplir.

Lors de la planification, j’ai immédiatement dû retenir quelles fonctionnalités qui allaient me prendre du temps pour implémenter. Ma vision ambitieuse de ce projet m’a amené à la réflexion lorsque j’ai dû faire face à mes premiers soucis de délais. Bien que les fonctionnalités demandées dans le cahier des charges aient été toutes réalisées, j’ai dû abandonner l’implémentation de quelques options supplémentaires comme la gestion des fichiers volumineux ou l’utilisation simultanée du programme pour laisser la place à des résolutions de bugs et l’amélioration de la stabilité générale de l’application.

Grâce à l’étape d’analyse, j’ai pu visualiser globalement à quoi ressemblerait l’application une fois terminée. J’ai élaboré des solutions pour toutes les implémentations dont la création semblait superflue et préparé au mieux l’environnement de travail avec lequel toute l’application serait façonnée.

C’est lors de l’étape de réalisation que j’ai pu mieux approfondir mes connaissances au niveau de l’utilisation du protocole IMAP, de la gestion d’une boite mail ou encore de la manipulation de fichiers dans un dossier Windows, le tout en langage C#. L’exploitation de librairies a changé la façon dont je visualise et conçois mon code. Ces « extensions » permettent une implémentation plus rapide de certaines fonctionnalités.

Pendant la phase de test, j’ai fait face à de nombreux problèmes et bugs qui ont parfois mis en danger le fonctionnement général de l’application. C’est à cause de ces problèmes rencontrés que certaines fonctionnalités supplémentaires ont été abandonnées au profit de la résolution de bugs et l’amélioration du programme.

Pour résumer, je dirais que ce projet m’a permis de mieux visualiser la charge de travail à accomplir par rapport à un cahier des charges donné et à mieux gérer mon temps pour la réalisation d’une application.

Divers

# Divers

## Journal de travail

### C:\Users\Marco\AppData\Local\Microsoft\Windows\INetCache\Content.Word\jdt1.pngSemaine 1

### Semaine 2

### Semaine 3

### Semaine 4

### Semaine 5

### Semaine 6

## Gantt final

### C:\Users\Marco\AppData\Local\Microsoft\Windows\INetCache\Content.Word\g1.pngSemaine 1

### Semaine 2

### Semaine 3

### Semaine 4

### Semaine 5

### Semaine 6

## Webographie

GitHub

« MailKit »

Jstedfast

<https://github.com/jstedfast/MailKit>

System.Data.SQLite

« SQLite »

<https://system.data.sqlite.org/index.html/doc/trunk/www/index.wiki>

Snipplr

« Base64 Encode or Decode a File »

Rengber

<http://snipplr.com/view/6034/base64-encode-or-decode-a-file/>

Haishi’s Blog

« Track a file when it’s modified, moved, or renamed under NTFS using C# »

Haishi

<http://haishibai.blogspot.ch/2010/08/track-file-when-its-modified-moved-or.html>

GitLab

« csharp-aes »

Petr Čech

<https://gitlab.com/czubehead/csharp-aes>

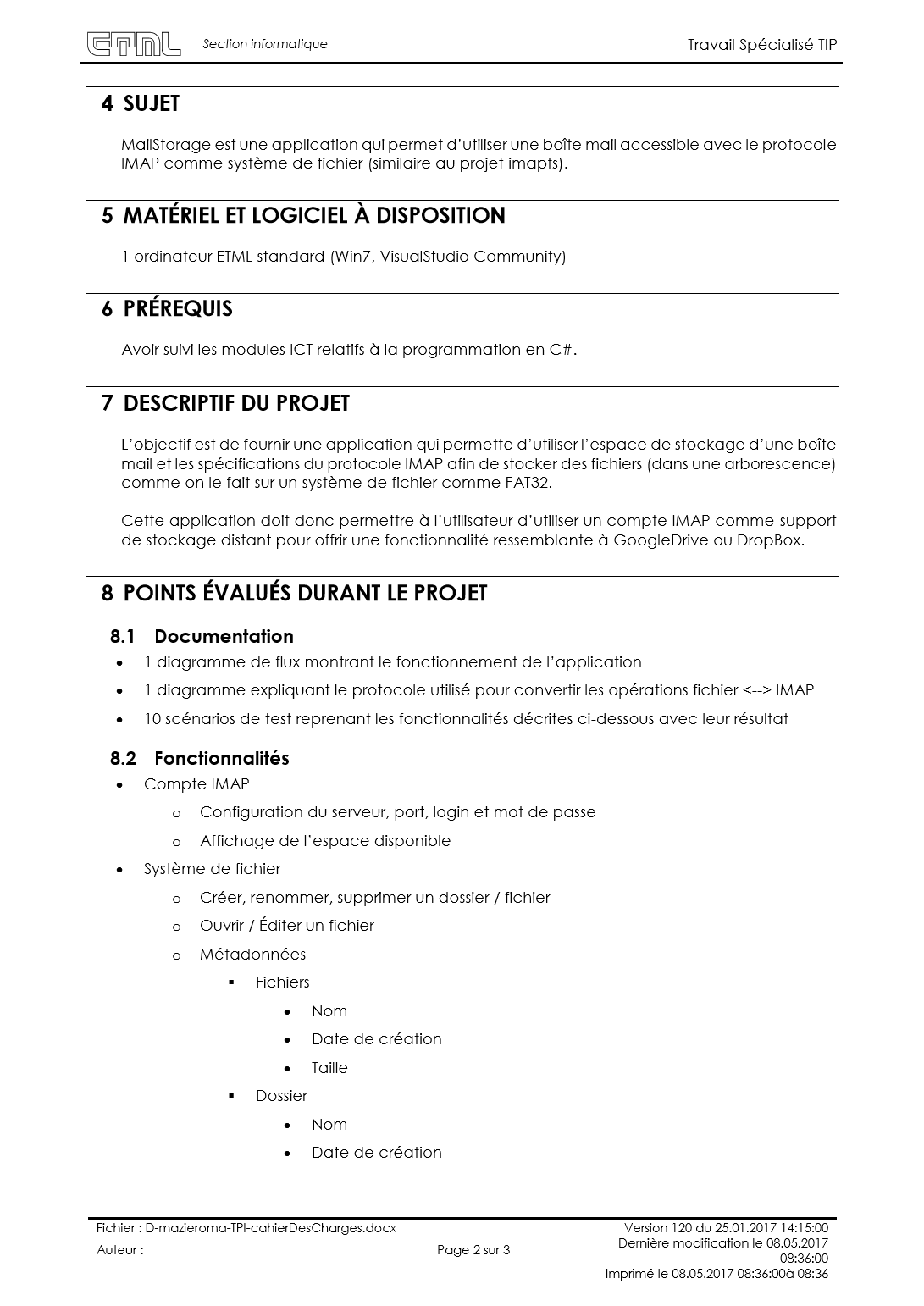
Stackoverflow

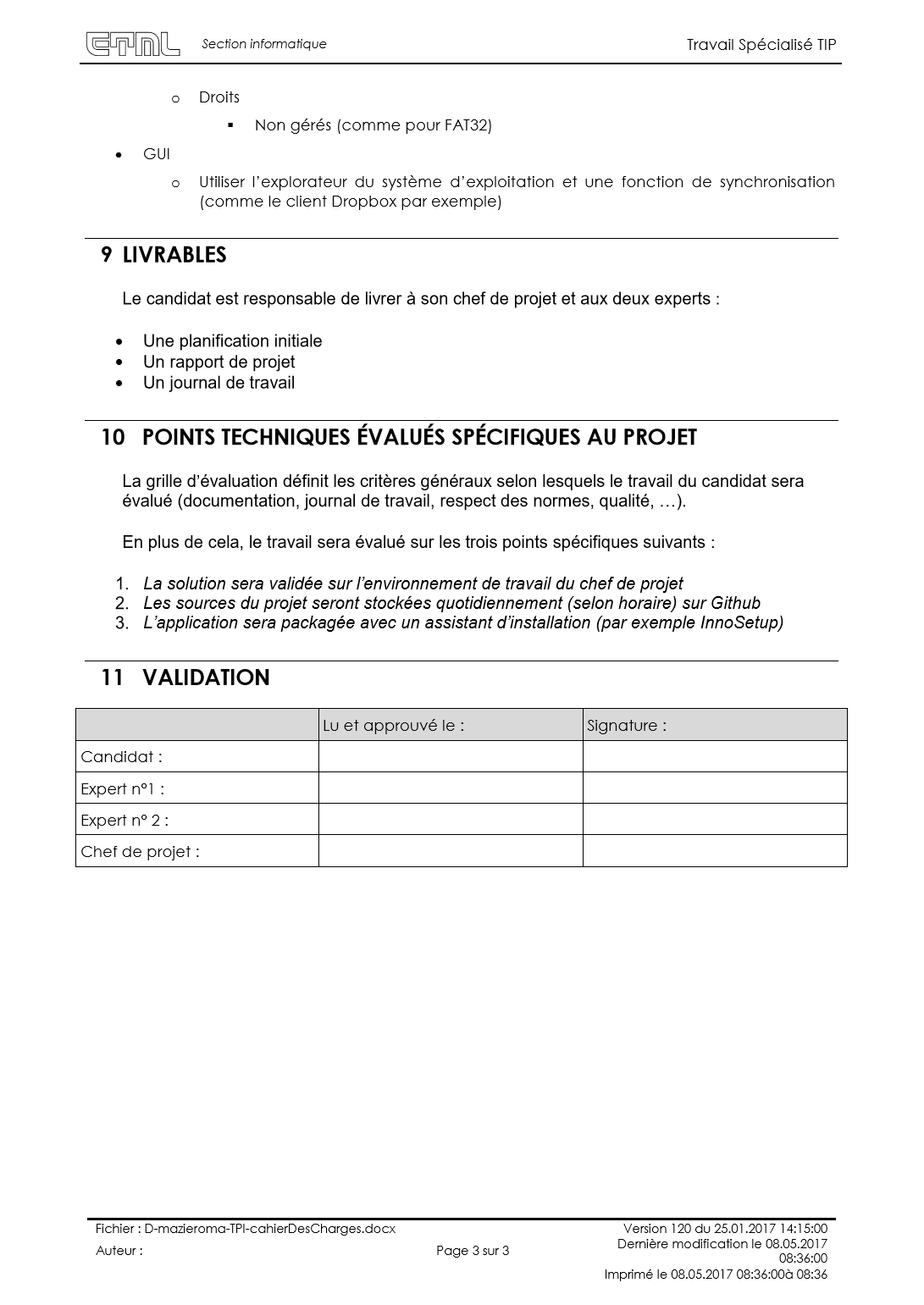
« How to asynchronously wait for x seconds and execute something then? »

Dennis G

https://stackoverflow.com/questions/9112305/how-to-asynchronously-wait-for-x-seconds-and-execute-something-then

## C:\Users\Marco\AppData\Local\Microsoft\Windows\INetCache\Content.Word\1.pngCahier des charges





1. Pour plus de détails, voir le cahier des charges en annexe [↑](#footnote-ref-1)
2. « Internet Message Access Protocol », <https://fr.wikipedia.org/wiki/Internet_Message_Access_Protocol>, 22.02.2017 [↑](#footnote-ref-2)
3. « Post Office Protocol », <https://fr.wikipedia.org/wiki/Post_Office_Protocol>, 02.05.2017 [↑](#footnote-ref-3)
4. « Browse for a directory in C# », <http://stackoverflow.com/questions/11767/browse-for-a-directory-in-c-sharp>, 08.08.2014 [↑](#footnote-ref-4)