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 4](#_Toc482353695)

[1.1 Titre 4](#_Toc482353696)

[1.2 Description 4](#_Toc482353697)

[1.3 Matériel et logiciels à disposition 4](#_Toc482353698)

[1.4 Prérequis 4](#_Toc482353699)

[2 Planification Initiale 5](#_Toc482353700)

[2.1 Liste des tâches 5](#_Toc482353701)

[2.2 Diagramme de Gantt 5](#_Toc482353702)

[3 Analyse 6](#_Toc482353703)

[3.1 Opportunités 6](#_Toc482353704)

[3.1.1 Utilisation du protocole IMAP 6](#_Toc482353705)

[3.1.2 Stockage de fichiers sur boite mail 7](#_Toc482353706)

[3.1.3 Approfondissement des connaissances 8](#_Toc482353707)

[3.2 Document d’analyse et conception 8](#_Toc482353708)

[3.2.1 Architecture du projet 8](#_Toc482353709)

[3.2.2 Interfaces 11](#_Toc482353710)

[3.2.3 Vérification du formulaire de connexion 13](#_Toc482353711)

[3.2.4 Connexion IMAP 15](#_Toc482353712)

[3.2.5 Interactions base de données 16](#_Toc482353713)

[3.2.6 Détection des modifications de fichiers 17](#_Toc482353714)

[3.2.7 Organisation des fichiers dans la boite mail 18](#_Toc482353715)

[3.2.8 Transformation des fichiers « dossier – boite mail » 18](#_Toc482353716)

[3.2.9 La synchronisation « dossier – boite mail » 19](#_Toc482353717)

[3.3 Conception des tests 20](#_Toc482353718)

[3.3.1 Scénario de connexion 20](#_Toc482353719)

[3.3.2 Scénario de synchronisation 20](#_Toc482353720)

[3.3.3 Scénario de changement de compte 20](#_Toc482353721)

[3.3.4 Scénario d’espace disponible 21](#_Toc482353722)

[3.3.5 Scénario de première synchronisation 21](#_Toc482353723)

[3.3.6 Scénario de création de fichier 21](#_Toc482353724)

[3.3.7 Scénario de suppression de mail 22](#_Toc482353725)

[3.3.8 Scénario de suppression de fichier 22](#_Toc482353726)

[3.3.9 Scenario d’édition de fichier 22](#_Toc482353727)

[3.3.10 Scenario de gestion des erreurs 22](#_Toc482353728)

[3.4 Planification détaillée 24](#_Toc482353729)

[3.4.1 Semaine 1 24](#_Toc482353730)

[3.4.2 Semaine 2 25](#_Toc482353731)

[3.4.3 Semaine 3 26](#_Toc482353732)

[3.4.4 Semaine 4 27](#_Toc482353733)

[3.4.5 Semaine 5 28](#_Toc482353734)

[3.4.6 Semaine 6 29](#_Toc482353735)

[4 Réalisation 30](#_Toc482353736)

[4.1 Dossier de Réalisation 30](#_Toc482353737)

[4.1.1 Interface utilisateur 30](#_Toc482353738)

[4.1.2 Connexion au serveur IMAP 34](#_Toc482353739)

[4.1.3 Interactions avec la base de données 34](#_Toc482353740)

[4.1.4 Interactions avec la boite mail 34](#_Toc482353741)

[4.2 Dossier des tests 34](#_Toc482353742)

[5 Conclusion 34](#_Toc482353743)

[5.1 Bilan des fonctionnalités demandées 34](#_Toc482353744)

[5.2 Bilan personnel 34](#_Toc482353745)

[6 Divers 34](#_Toc482353746)

[6.1 Journal de travail 34](#_Toc482353747)

[6.2 Webographie 34](#_Toc482353748)

[6.3 Cahier des charges 35](#_Toc482353749)

# 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, évènementielle, orientée objet et en gestion de projet. Toutes ces compétences ont été acquises lors des cours à l’ETML.

# 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 est basée sur cette planification initiale.

# 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)

Crée 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 ou 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 peuvent contiennent généralement un sujet, un contenu, une adresse mail de l’émetteur, une adresse du destinataire et parfois, le mail peut aussi être accompagné d’une pièce jointe.

Mail

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

**A :** [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.

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. Le chemin d’accès au fichier peut être stocké dans le sujet du message, ainsi il reste toujours une trace du nom et de l’extension du fichier.

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

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

**Sujet :** C:\Users\name\file.txt

**Contenu :** 5pWt

Mail

5pWt

Fichier en données Base64

Pour la récupération des fichiers, il suffit simplement de récupérer

### 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 de 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 pour communiquer en IMAP avec la boite mail
* Une classe pour communiquer avec la base de données
* Une classe pour gérer les fichiers

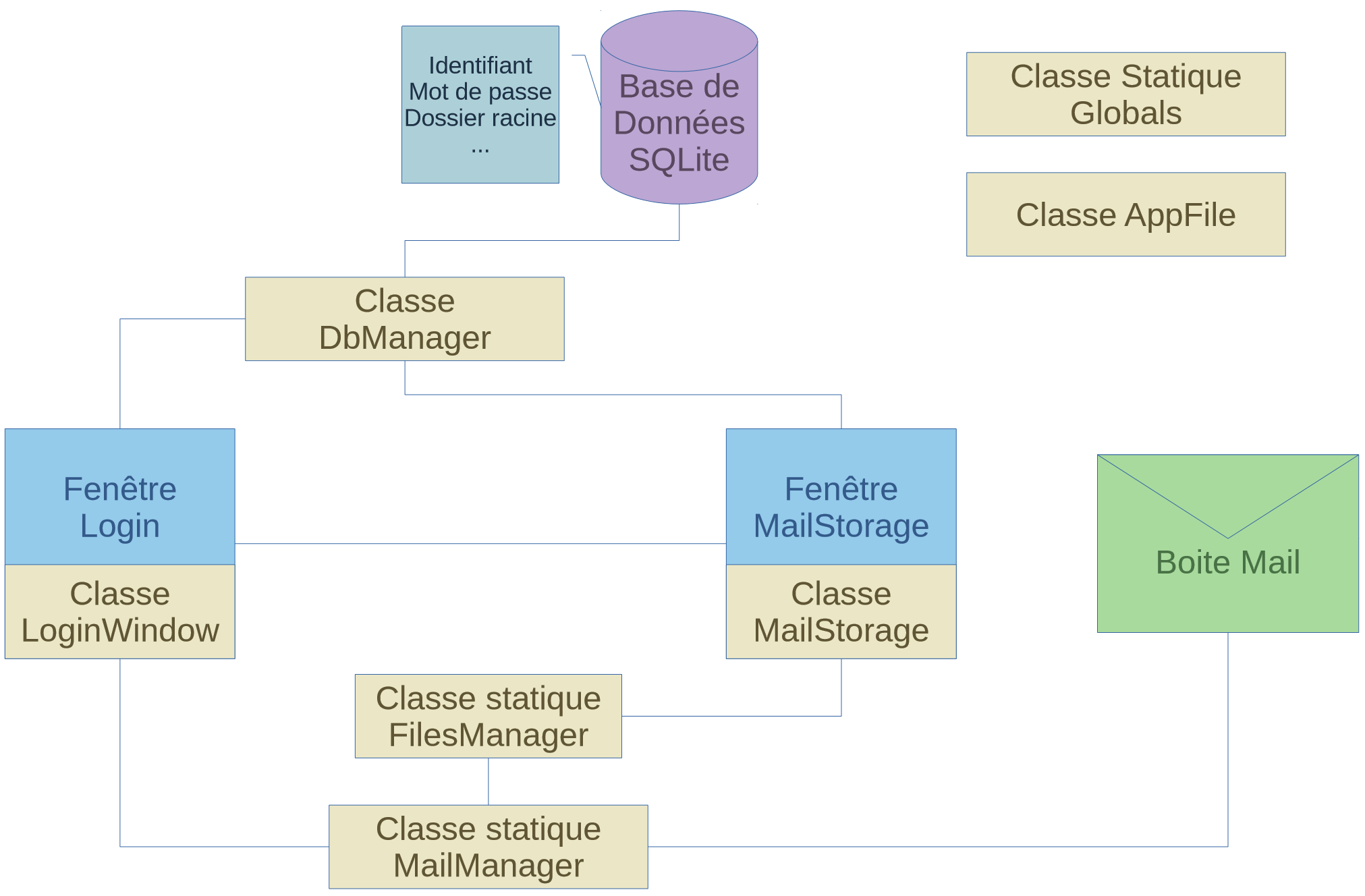
Les classes principales sont celles qui gèrent les fenêtres, c’est depuis ces éléments que les autres classes sont instanciées et appelées.

Figure : 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 réception 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 est ensuite placé 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, identifiant, 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 : Fenêtre de connexion

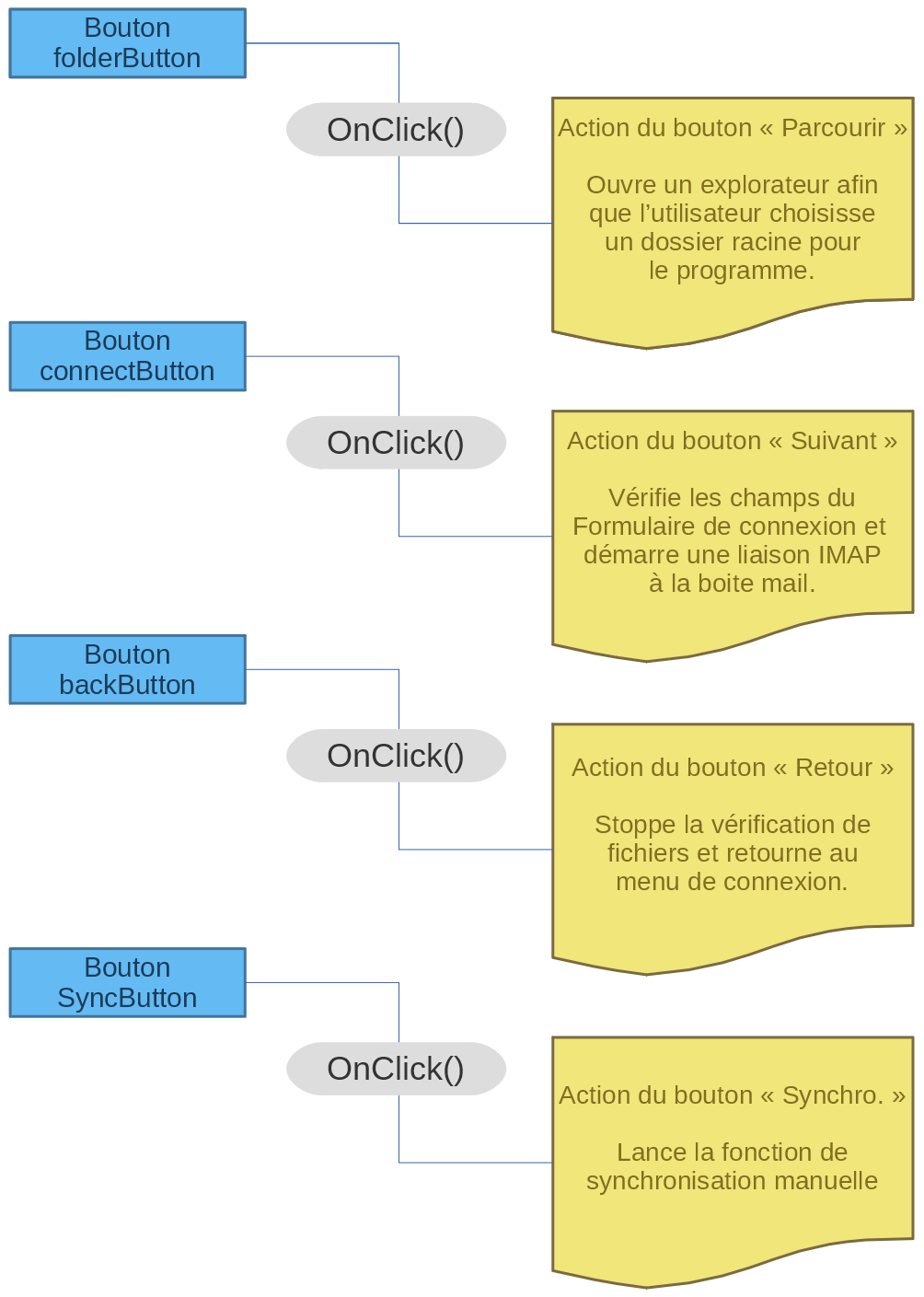


Figure : Fenêtre de traitement des fichiers

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

Les interactions avec l’interface ne sont pas très nombreuses. Ce ne sont que des boutons qui réagissent au clic. Le schéma évenementiel ci-dessous décrit les interractions 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

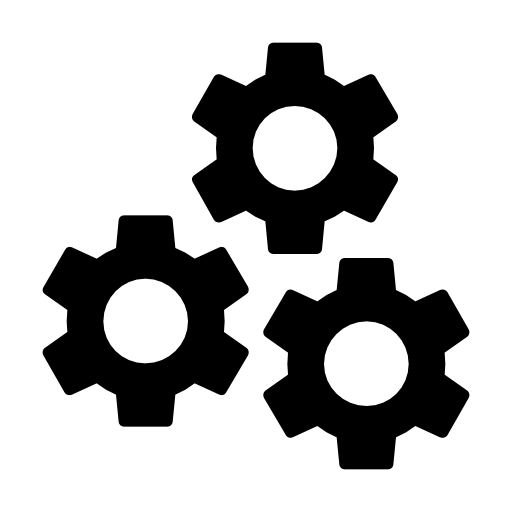
La première étape lors de l’utilisation du programme est la connexion en IMAP au serveur mail. Pour ce faire, l’utilisateur doit entrer plusieurs données dans un petit formulaire afin que le programme ait les valeurs nécessaires à la connexion.



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 ou 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 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 corresponde 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.

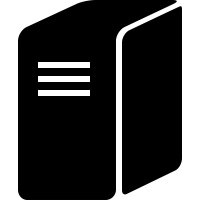
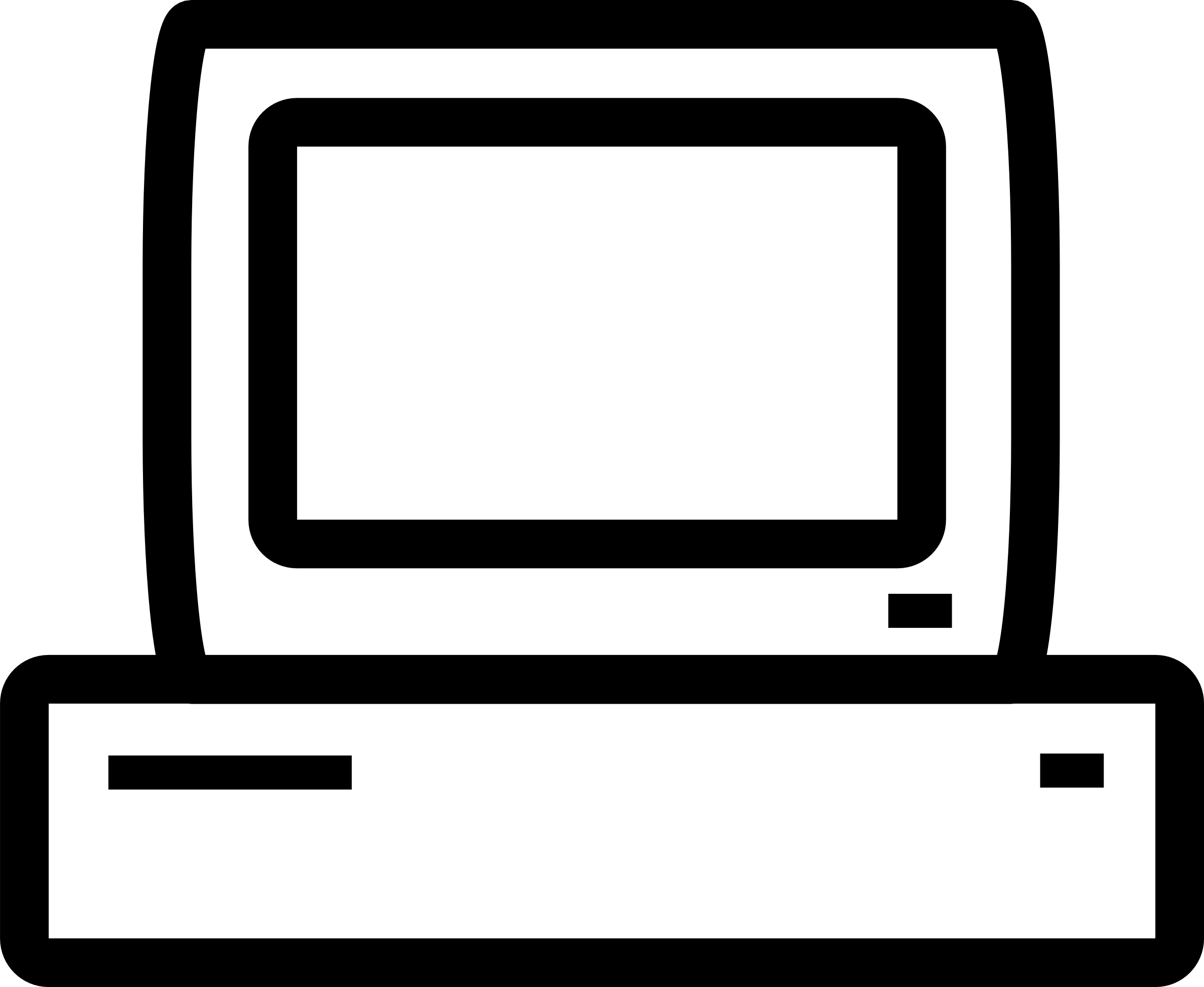
Dans ce champ, l’utilisateur ne peut rentrer des données manuellement. Pour remplir le champ il faut cliquer sur le bouton « Parcourir » afin qu’une fenêtre d’explorateur Windows s’ouvre pour que l’utilisateur puisse choisir le dossier racine qu’il souhaite.

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.

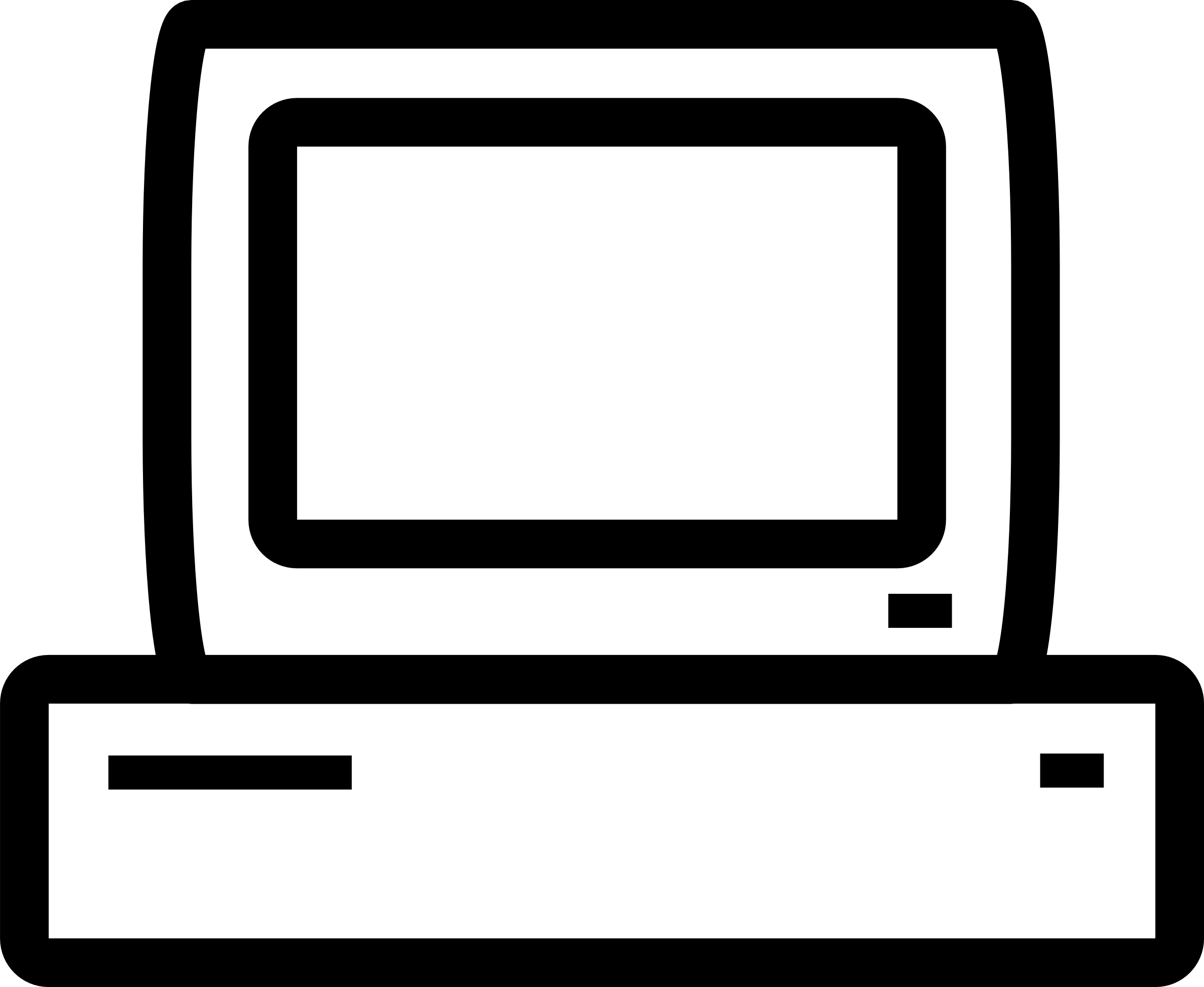
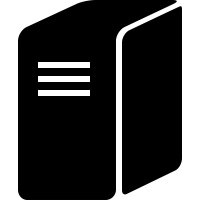
### Connexion IMAP

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

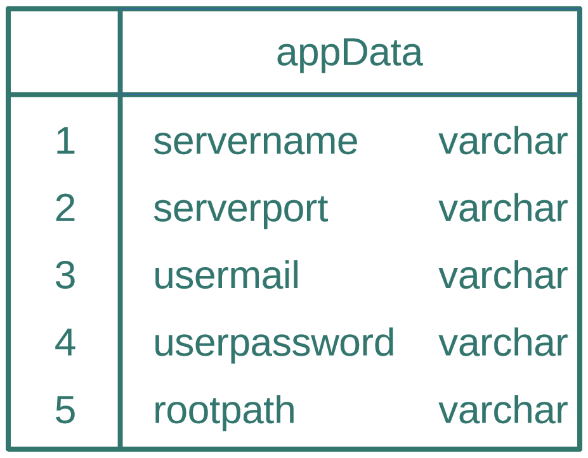
Ensuite, une demande d’authentification est lancée avec les identifiants de l’utilisateur (adresse mail et mot de passe). Le serveur répond soit avec un OK soit avec un refus.

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.

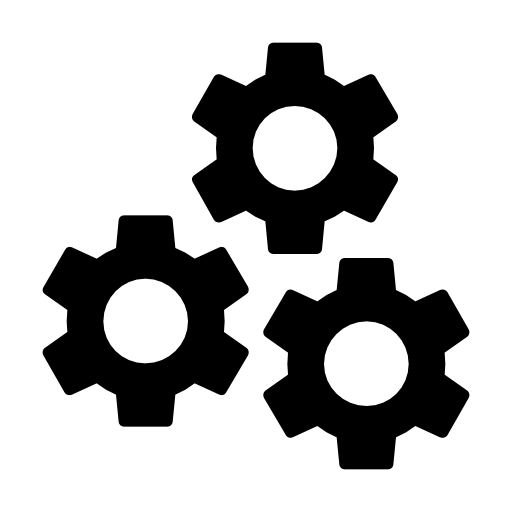
La table appData est composée de 5 champs. Tout d’abord, les deux champs concernant le nom et le port du serveur IMAP.

Ensuite viennent les champs de l’utilisateur qui stockent son adresse mail et son mot de passe 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

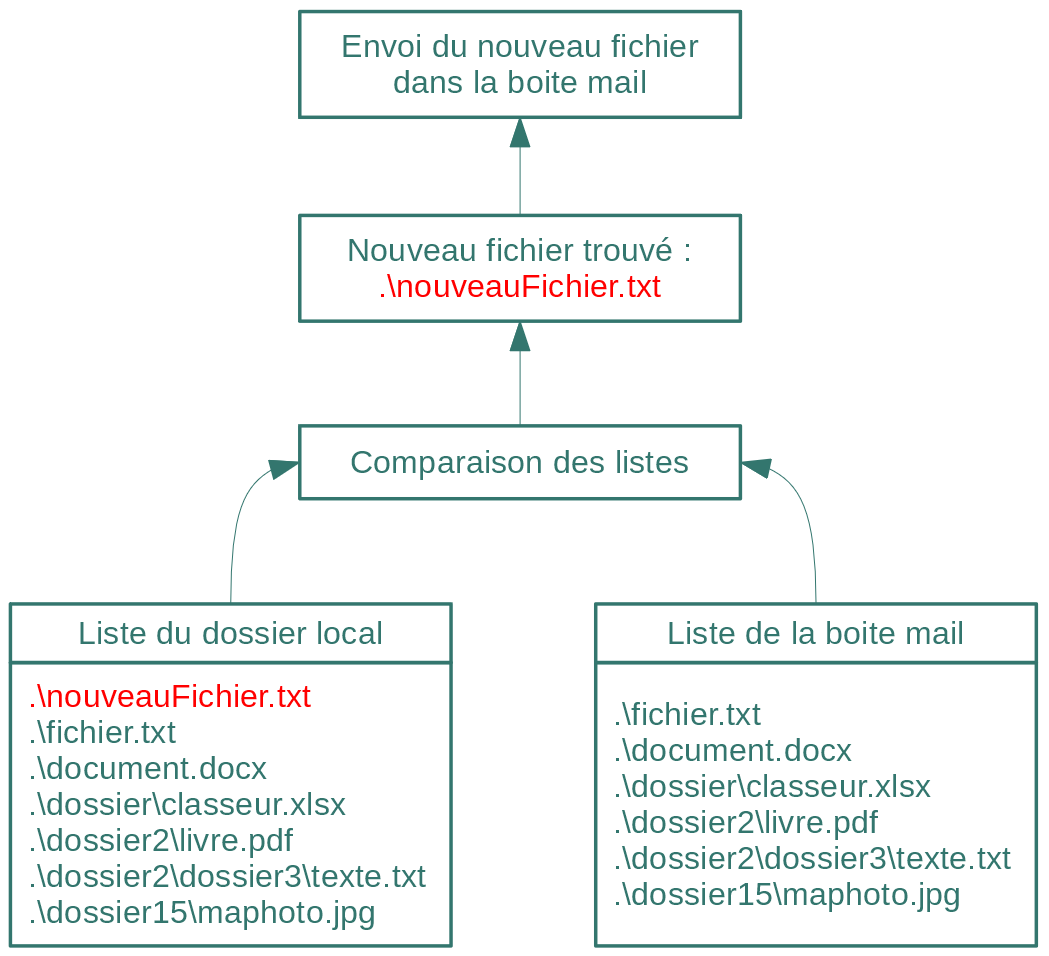
Base de données

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, ce qui permet de connaitre l’arborescence totale du dossier racine.

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 fichié 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 l’emplacement du fichier dans le dossier local 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

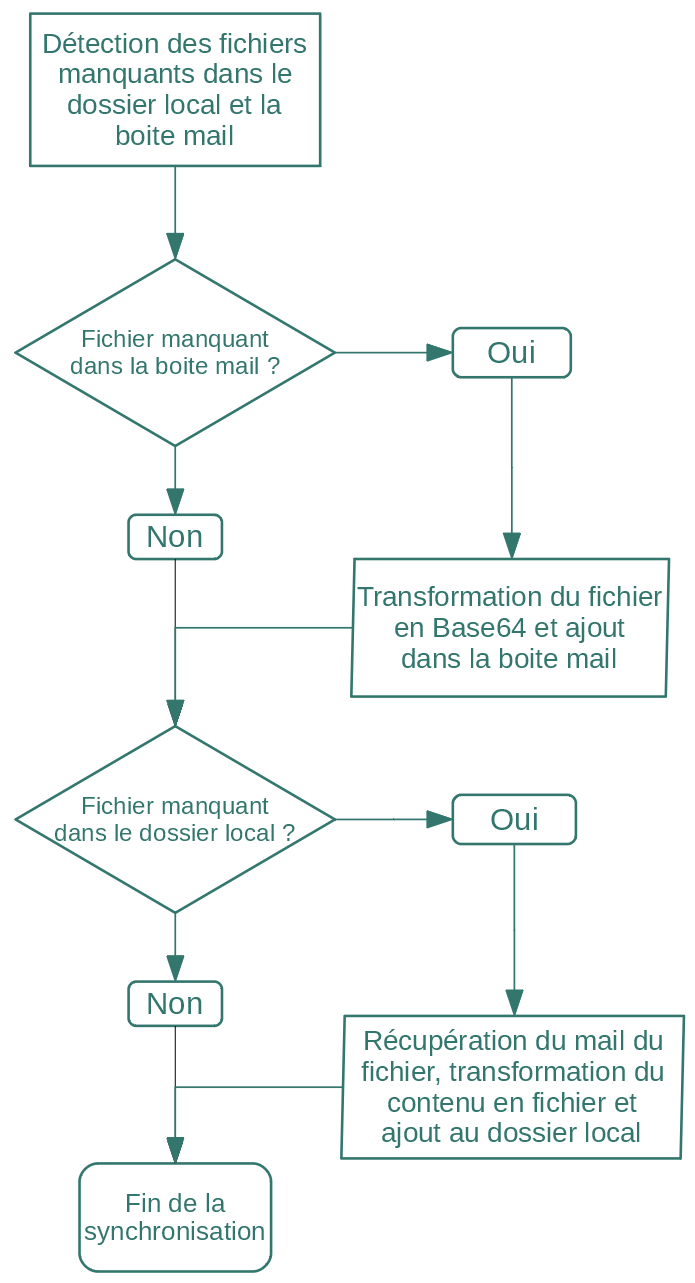
Ce texte est ensuite placé dans la boite mail, dans le contenu d’un nouveau mail situé dans le dossier « MailStorage ». Le chemin d’accès au fichier est quant à lui placé dans le sujet du mail.

### La synchronisation « dossier – boite mail »

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.

Puis, si au contraire, il manque des fichiers dans le dossier local, les mails contenant les fichiers sont lus un par un et leur contenu est récupéré et transformé en fichier qui sera placé dans le dossier local.

## Conception des tests

### Scénario de 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 de synchronisation

Le test de vérification du bouton synchroniser s’effectue afin de vérifier que la synchronisation « dossier local – boite mail » se fasse 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 de 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 au préalable 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 d’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 de première synchronisation

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 de création de fichier

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 de 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 de 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 |

### Scenario d’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 | Edition d’un fichier dans le dossier local |
| Résultat | Mise à jour automatique du fichier dans la boite mail |

### Scenario de 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

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 crées 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 contructeur de la calsse 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() - Evénement |
| Paramètres | **Object** sender  **EventArgs** e |
| Retour | - |
| Description détaillée | Cette fonction est apellée 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() - Evénement |
| 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 | ValidateConnection() - Evénement |
| 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’ |

|  |  |
| --- | --- |
| 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 é la base de données est effectué.  Une vérification est ensuite faite dans la base de données afin de récupérer des données existantes si il y en a.  Dans le cas ou 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() - Evénement |
| Paramètres | **Object** sender  **EventArgs** e |
| Retour | - |
| Description détaillée | Appelée quand le bouton « Parcourir » est cliqué.  Cette fonction affiche simplement un fenêtre d’explorateur afin que l’utilisateur choisisse son dossier racine. |

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

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

### 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. Toutes les interactions avec le serveur IMAP et la boite mail se font depuis la classe *MailManager*.

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.

|  |  |
| --- | --- |
| 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 un 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 crée. |

### 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.

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 crée 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. |

## Dossier des tests

# Conclusion

## Bilan des fonctionnalités demandées

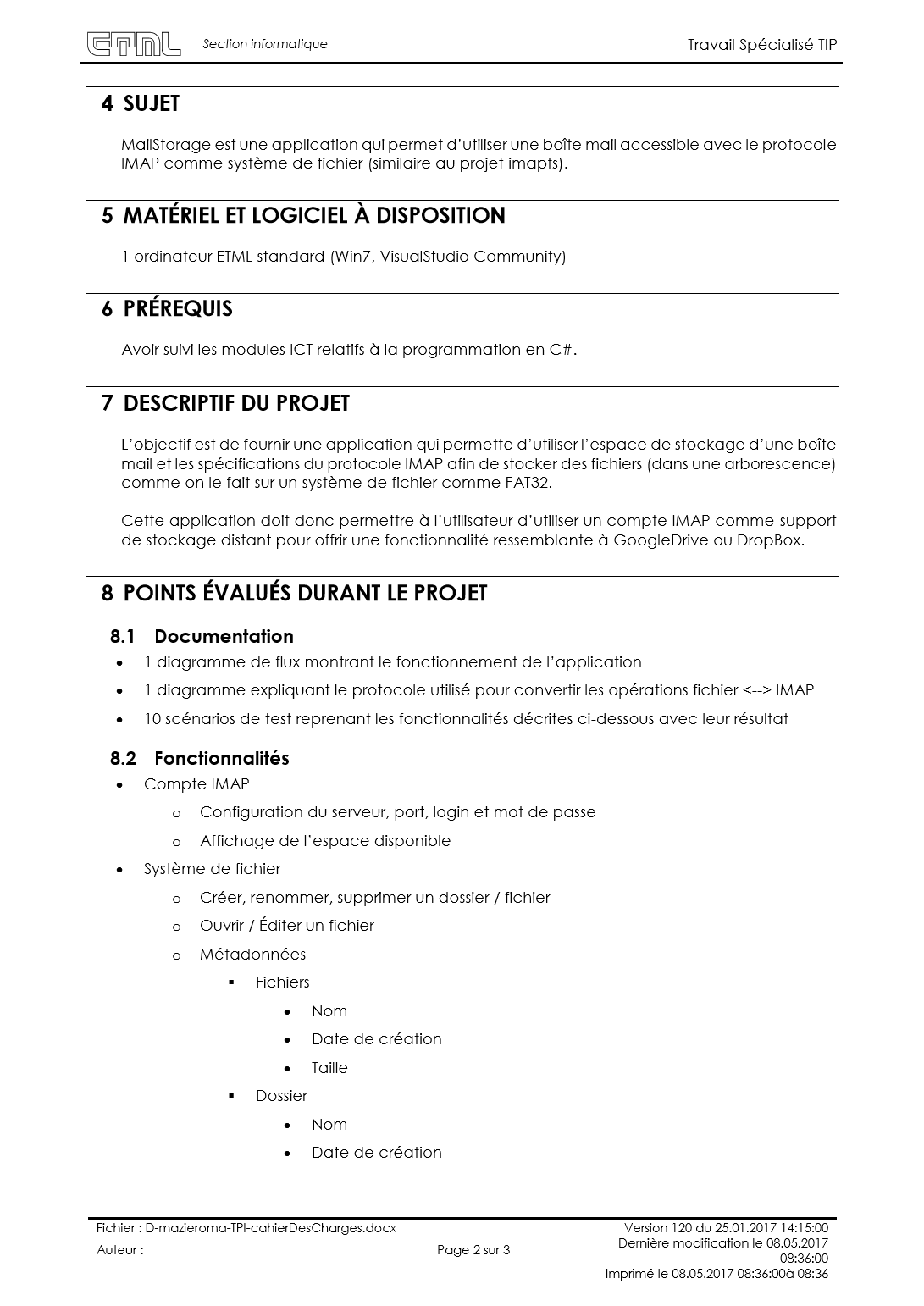
## Bilan personnel

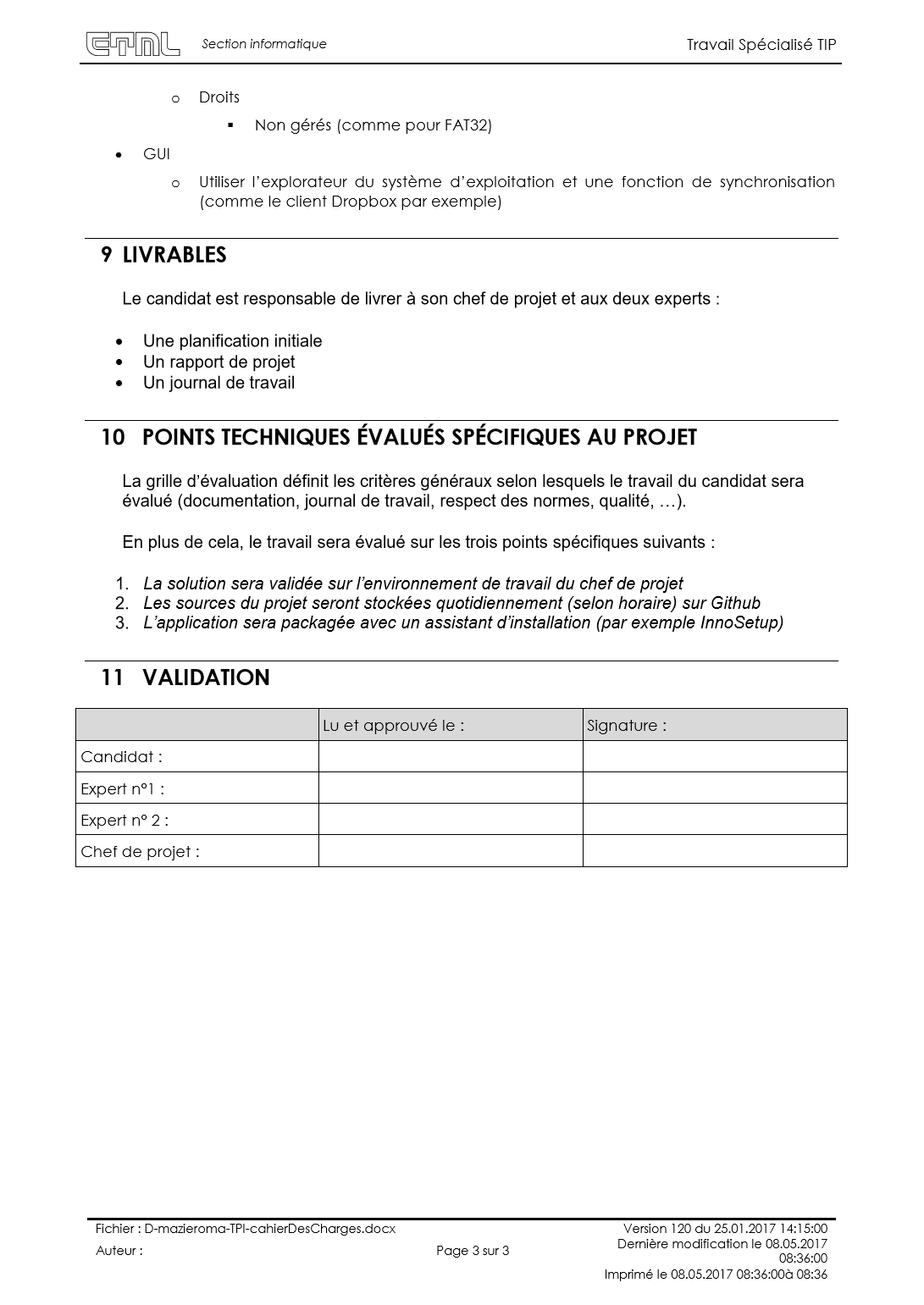
# Divers

## Journal de travail

## Webographie

## 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)