Rapport
"Créations de pages web mensuelles”

Plan:

I. Présentation

1. Résumé du projet

Le but était de créer une interface en Python pour rassembler des informations d'un fichier excel multi feuilles à passer en "csv" pour un traitement automatique.
Puis en faire une page HTML (HTML, CSS) envoyée par mail et une page web utilisée sur un site internet. Les paramètres de l'interface sont la liste des feuilles, la liste des rubriques, le mois et l'année.

Ci-dessous, le lien vers la page de la forge décrivant le projet :

https://forge.info.unicaen.fr/projects/projets-l3/wiki/Cr%C3%A9ation_de_pages_web_mensuelles

2. Langages utilisés

a. Python

Pour ce projet nous avons principalement utilisé python, version 3.7.

b. HTML

Pour tout ce qui est de l'affichage HTML, nous avons codé un générateur de pages en python, avec les balises HTML. Pour ce qui est du CSS, nous avons récupéré le CSS fourni dans l'exemple de page web.

c. Modules (bibliothèques)

Nous avons utilisé plusieurs bibliothèques intégrées et non intégrées à Python :

```
-tkinter     #interface graphique
-os          #gestion des fichier
-shutil      #gestion de fichier
-datetime    #gestion du temps
-pandas      #gestion des fichier xlsx
-numpy       #gestion des listes
-smtplib     #envoi de mail
```

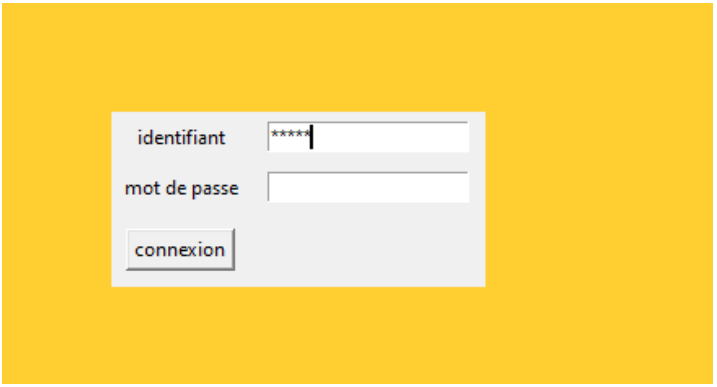
II. Documentation utilisateur

1. Utilisateur (schéma des infos)

La prise en main est relativement facile et efficace.
Au lancement, nous avons une interface de connexion pour adapter l'application à l'utilisation que l'utilisateur va avoir du programme (client ou administrateur).

À noter :

Pour le moment, l'application client ne se connecte pas à la base de données. Donc pour pouvoir se connecter en tant qu'utilisateur normal, il faut uniquement rentrer "salut" dans l'identifiant.
N'ayant pas de base de données, il n'y a pas d'interface graphique pour un administrateur, mais nous verrons qu'il n'y en a pas besoin pour une utilisation complète.



L'utilisateur rentre donc "salut" puis il clique sur connexion.

Image: Exemple de connexion

Une fois sur l'interface graphique, on a accès à plusieurs boutons et une interface d'affichage vide.
Pour pouvoir remplir cette interface, l'utilisateur doit cliquer sur le bouton "Ajouter un fichier xlsx" ce qui redirigera l'utilisateur vers une fenêtre de sélection de fichier à la racine de son ordinateur.

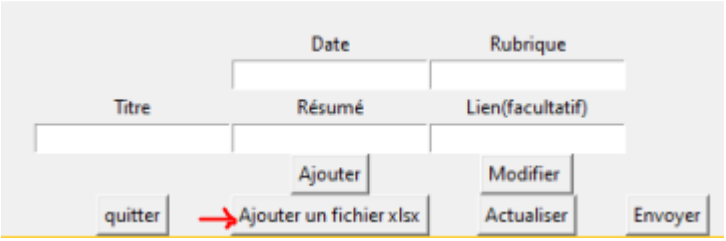
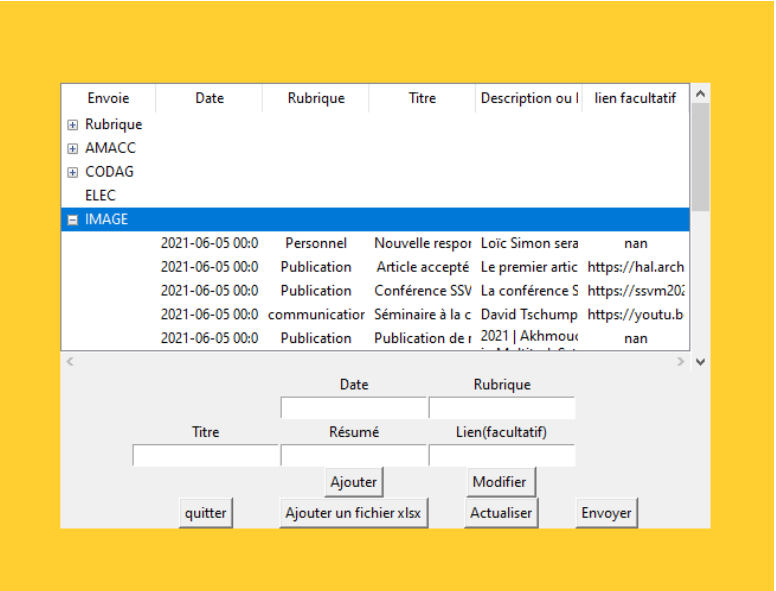


Image: Bouton d'ajout de fichier xlsx

Une fois le fichier xlsx sélectionné, l'interface se remplit des informations contenues à l'intérieur du fichier.
Mais pour pouvoir afficher les informations il faudra cliquer sur le bouton actualiser.

Le fichier xlsx doit exactement avoir le format du fichier exemple qui nous a été donné.
(avec une feuille rubrique, avec pour première colonne toutes les rubriques utilisées)
Un nombre indéterminé de feuilles de groupe de recherche contenant un nombre indéterminé de lignes, mais exactement 5 colonnes
(Date, Rubrique, Titre, Description ou lien, Lien facultatif)



Ainsi grâce au bouton mis à disposition:

- Ajouter (en cliquant sur une rubrique) permet d'ajouter un élément dans les listes.
- Modifier, comme son nom l'indique, permet de modifier un élément de la liste.
- Envoyer enverra une copie de tous les éléments visibles du tableau au serveur afin de les transformer en page web. ("Rubrique" ne sera pas affiché sur la page html.)
- Bouton quitter qui permet de quitter la page.

2. Administrateur

Pour l'administrateur, nous avons eu comme consigne qu'il pouvait uniquement créer de nouvelles rubriques. Nous avons décidé qu'il n'y avait pas besoin de créer une interface particulière pour lui, car la simple modification sur fichier xlsx en ajoutant des rubriques dans la liste Rubrique, rajouterai des rubriques au serveur.
Cependant, nous avons eu des idées d'actions que seul un administrateur peut réaliser. C'est pourquoi le code prend en compte la possibilité d'avoir un administrateur.
Nous en reparlerons dans la partie développeur et Perspective (améliorations).

3. Développeur

Cette partie sera la plus technique. Elle explique le fonctionnement général des fonctions utilisées dans `coté_client` et dans `coté_server`.
À noter que le coté_server n'est pas autonome comme le `coté_client`. Le `coté_server` a besoin (pour le moment) de l'intervention d'un informaticien pour qu'il fonctionne.

a. Côté utilisateur

- **Main**

La partie client est composée de 2 principaux composants : `main.py` et `UserPage.py`

main.py initialise la page fenêtre initiale qui nous permettra d'afficher les informations que l'on veut.
Sa classe, appelée MainClass, n'utilise que le module Tkinter.
Elle est composée d'un __init__ qui crée une première frame et initie son contenu avec l'appel de l'affichage de la classe Connexion.

La seule méthode de MainClass est switch. Elle a pour paramètre la nouvelle frame que l'on veut afficher. Elle permet d'intégralement effacer la Frame active, et d'initialiser une nouvelle Frame avec de nouvelles informations.

- **Connexion**

Connexion.py possède un __init__ mettant en place son affichage et les boutons, ainsi qu'une méthode testLogin() permettant de tester les identifiants et mot de passe que l'utilisateur saisit.
Le serveur répond alors True ou False, True avec un String classant les droits de l'utilisateur en SimpleUsers et SuperUtilisateur pour les administrateurs, sinon False et affiche à l'écran une erreur pour que l'utilisateur saisisse à nouveau ses identifiants de connexion.

- **UserPage**

UserPage.py est la seconde classe la plus importante pour l'utilisateur.
Elle permet l'affichage des informations ainsi que le traitement des données des fichiers xlsx.
Elle regroupe l'utilisateur des fonctions et classes annexes de tous les autres fichiers python (FenAjoute.py, LectureCSV.py, suppression.py)

La class UserPage est donc composée de plusieurs méthodes. Elle n'a pas besoin de paramètres lors de son appel.
Dès que cette classe est appelée, elle met en place 2 frames différentes.
La première Frame est composée de la Frame initialisée lors de l'appel de UserPage, depuis la méthode switch du main.py, et prend en compte le tableau d'affichage fait grâce à Treeview.
La seconde Frame regroupe tous les boutons et les zones de saisie.
Ainsi chaque bouton de cette seconde frame permettra l'appel de méthodes.

button_Add permet d'appeler la classe Recherche contenue dans le fichier FenAjout.py.
Cette dernière initie la recherche de fichier en xlsx, puis l'appel de la méthode ConvertXlsxToCsv qui créera une copie du fichier sélectionné dans un dossier temporaire.

button_Actualise permet d'appeler la méthode affichage() sans paramètres nécessaires.
Cette méthode appelle la fonction lecture_feuilles() du fichier lectureCSV.py.
Cette dernière renvoie une liste de toutes les informations contenues dans le fichier xlsx par feuilles.
Spécification de la liste :

- Liste en 4 dimensions [][][][].
- Le premier [] correspond à la feuille sélectionnée, le second [] peut prendre uniquement 2 paramètres : 0 ou 1.

Si le paramètre est de 0, alors on obtient le nom de la feuille.

Si l'on rentre 1 on a accès aux informations contenues dans la feuille. On peut alors aller plus loin dans la liste et en rajouter [][].
Représentation : [[element,[[],[],[]]],[element,[[],[],[]]],[element,[[],[],[]]]]

Puis (toujours dans la méthode affichage), on créer 2 listes différentes qui vont être utilisées dans la classe self.data_parent qui contient le nom de toutes les feuilles. Ces feuilles seront dans le Treeview les parents des informations affichées.
Enfin, une liste self.data qui contient toutes les informations des feuilles xlsx.

Puis une fois que ces 2 listes sont créées, on insère leur contenu dans self.tab (pour tableau) sous la forme de parent/fils,fils,... ce qui formera notre Treeview.

Les Bouton Ajouter (input_button) et Modifier (edit_button) ont un comportement similaire.

input_button appelle la méthode ajoute().
Cette méthode vérifie en premier lieu qu'un parent a été sélectionné. /!\ (la liste des parents du Treeview commence a n+1 ! soit 0+1 pour le premier élément du tableau. Car le paramètre 0 ou iid=0 est utilisé initialement par Treeview pour stocker la liste de tous les parents)
Une fois une rubrique sélectionnée (un parent), on récupère simplement les informations des zones d'écriture (Entry) et on les insère à tab.

edit_button, quant à lui, appelle la méthode modifier() qui se charge de vérifier si l'on a bien sélectionné un élément fils, et le modifie grâce à la fonction focus() de Treeview, qui permet de sélectionner rapidement l'iid d'un fils sans avoir à le chercher.

Le dernier bouton est Envoyer (button_Envoyer). Celui-ci appelle la méthode écriture().
Initialement cette fonction devait uniquement récupérer les informations du Treeview et les envoyer dans les fichiers temporaires afin qu'ils soient envoyés au serveur.
La partie réseaux n'étant pas encore faite, elle envoie directement les fichiers dans cote_server/fichier_temporaires.

	Data
0	2021-06-0
1	Publication
2	Survey pub
3	Paul Dorbe
4	https://link

	Data
0	2021-06-0
1	thèse/HDR
2	Nouveau d
3	Léo Paviet
4	nan

Cette fonction écriture fonctionne de la sorte. Nous faisons une boucle pour chaque élément parent que nous avons. Nous pouvons aussi bien utiliser la liste des noms parents que les parents treeview (ce sont les mêmes).
Puis nous créons des fichiers xlsx avec le nom de leurs parents, pour y mettre toutes les informations des feuilles initiales.
L'organisation des ces fichiers est simple. Chaque fichier a le nom d'un groupe de recherche, et toutes les informations sont rangées dans la colonne B du tableur.
Les informations sont alors rangées 6 par 6, car la librairie pandas écrit les informations sous forme de colonne ce qui facilitera la lecture du serveur.

Enfin, rapidement le fichier suppression.py permet de réinitialiser le fichier temp (temporaire), pour que le client n'ait pas de problèmes d'application qui se mette à prendre trop de place. Cette fonction est appelée à chaque ouverture d'un nouveau fichier xlsx pour traitement.

b. Côté serveur

Pour le moment, et comme dit plus haut, le serveur ne fonctionne pas tout seul. Cependant, toutes les fonctions nécessaires à son fonctionnement sont là. Il ne peut simplement pas recevoir d'information.
(La partie reliant le client au serveur est manquante, c'est seulement avec quelques semaines de plus qu'on aurait pu avoir une application fonctionnant a 100%)
Des recherches de communication en C ont été faites à partir de nos connaissances en réseaux, mais n'étant pas concluantes, nous ne les avons pas mises dans le rendu.

Le serveur s'organise autour de la création de compte-rendu HTML.
Les explications seront rapides car le fonctionnement de chaque partie reste relativement simple.
Le serveur s'axe principalement sur le fichier creation_header_style.py.
Initialement, il était prévu qu'il ne fasse que la génération du header et du style.
Cependant, comme il était plus simple de regrouper toute la transformation, ce fichier générera l'entièreté d'une page html (en appelant, bien sûr, des fonctions tierces).

Normalement, une fois que les fichiers xlsx sont arrivés dans les fichiers temporaires, la mise en route de création des fichiers HTML se fait par l'appel de creation_header_style.py.
(Cette opération est donc simulée pour le moment par l'appel de la fonction avec les paramètres "date" et "destination").
"destination" permet de savoir si le but est de générer un fichier HTML pour le serveur ou de l'envoyer par mail. Car si l'on l'envoie par mail, il faut que le Style.css soit intégré à la page HTML.

À son appel, elle commence par générer un fichier dans le serveur_Web/html avec pour nom la date du rapport mensuel au format "mois_année".

On y incorpore par write le header, les métadonnées ainsi que le fichier de style quand on envoie à un client.

Puis on appelle la fonction GenerationHTMLCSS() avec pour paramètre la date.
Elle va récupérer une liste des rubriques ainsi que les données des différents fichiers xlsx grâce à la fonction lecture_feuilles() qui s'occupe de retourner une liste des éléments complète (nom du groupe de recherche, date, type de projet, nom du projet, description et lien facultatif).
Ainsi grâce à cette liste, GenerationHTMLCSS() génère l'entièreté du contenu du fichier HTML.

<u>À noter :</u>

Pour pouvoir afficher les pages générées, nous avons mis en place un petit site web dans fichier_python/serveur_Web.
Son fonctionnement est très simple, il possède un fichier server.py qui une fois lancé émule le serveur web et affiche une liste de tous les fichiers HTML enregistrés dans html, grâce à index.html. Pour son fonctionnement il suffit de lancer server.py et le serveur marche.

Ainsi, il y a un fichier index.html qui regroupe les fichiers HTML.
Il est uniquement constitué de listes <ul>.
Ce fichier est généré en écriture écrasante afin de le re-générer à chaque fois qu'une nouvelle page HTML est faite. (Nous sommes toujours dans creation_header_style.py)

La réactualisation du fichier index est faite par le main du fichier Generationindex.py

Puis pour l'archivage, on crée les fichiers csv de chaque feuille que l'on a reçue. Ces fichiers csv sont stockés dans fichier_csv. (fonction Conversion du fichier converio_csn)

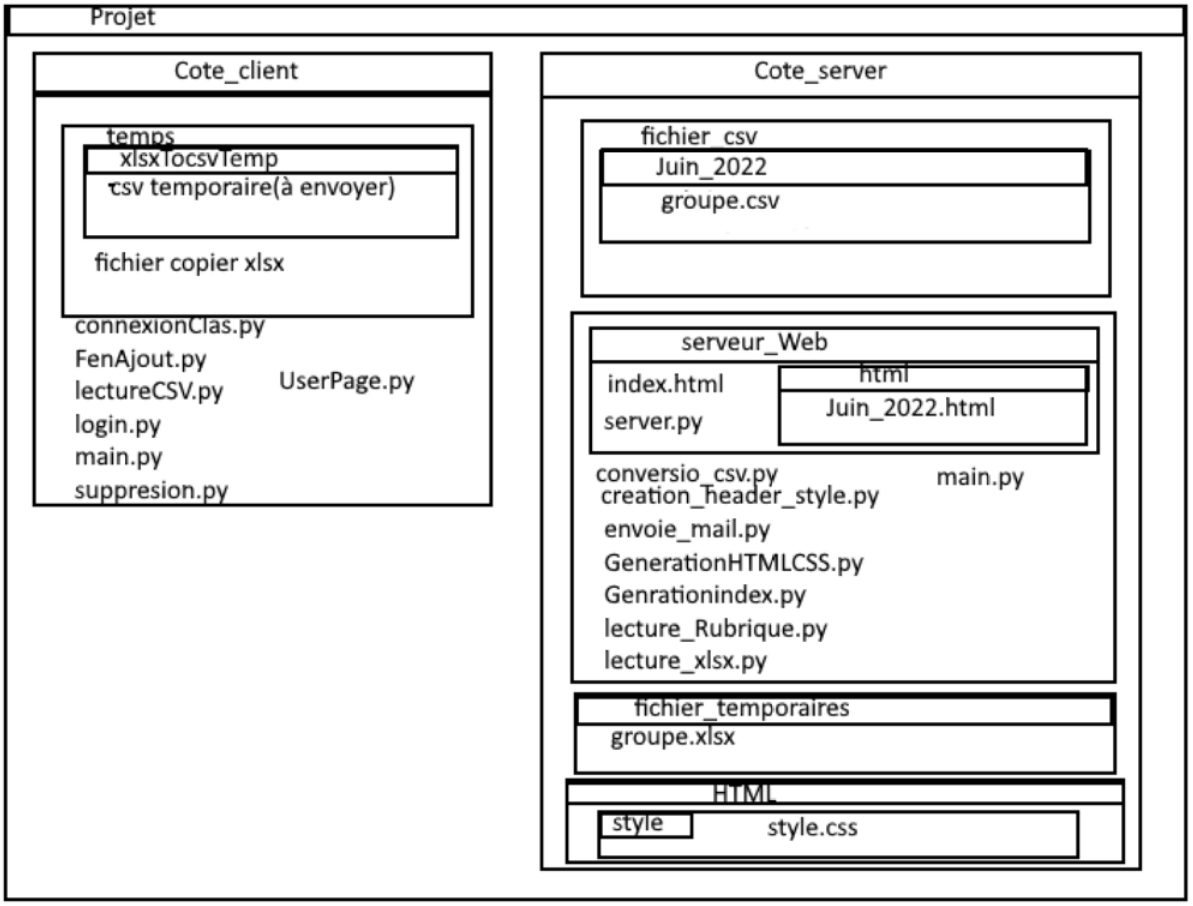
Enfin on supprime tous les éléments du fichier temporaire qui sont inutiles.

On peut remarquer qu'il y a un fichier envoie_mail.py résultat de notre recherche sur la manière d'envoyer le fichier HTML généré par email.
Le contenue de ce fichier est entièrement basé sur celui du site suivant :
https://latutotheque.fr/charlometre/tutoriels/informatique/comment-envoyer-un-email-avec-python
Adapté à nos besoins, ce programme envoie un fichier à l'adresse mail précisée.
Cependant il n'envoie qu'un fichier encodé en hexadécimal, que l'on n'est pas parvenu à re-transformer en binaire pour avoir un HTML utilisable.
Si l'on ouvre dans un éditeur de texte comme Atom le fichier reçu, on retrouve tout l'HTML que l'on voulait.

III. Documentation technique

1. Organisation des fichiers

Voici la représentation de l'organisation de nos différents fichiers et dossiers :



2. Plan des tableurs xlsx / csv

Le tableur initial est comme celui que l'on nous a donné pour exemple.
Composé :

- d'une feuille avec la date
- d'une feuille nommée Rubrique avec la liste des rubriques
- de plusieurs feuilles constituées de 5 colonnes avec une entête (Date, Rubrique, Titre, Description ou lien, Lien img/vidéo), avec un nombre indéterminé de lignes remplies d'informations.

Image: exemple de l'organisation des feuilles du fichier xlsx entrant

A
Rubrique
Personnel
Séminaire
Publication
communication
Projet
thèse/HDR
Offres d'emploi
presse
valorisation
Evènement de cohésion
focus personne évènement

Ci-contre : ce à quoi doit ressembler la liste des rubriques

Ci-dessous : représentation de la composition d'une feuille

	A	B	C	D	E
1	**Date**	**Rubrique**	**Titre**	**Description ou lien**	**lien img/ vidéo**
2	juin 2021	Publication	Publication dans une prestigieuse revue sur la sécurité 5G IoT	Un article paru dans la revue IEEE Networks (impact factor 8.80) de Lyes Khoukhi : "Prediction and Detection of FDIA and DDoS Attacks in 5G Enabled IoT"	DOI: 10.1109/MNET.011.2000449
3	août 2021	valorisation	Co-Chair of conference	Marc Spagnol: Co-Chair of APWeb-WAIM 2021 (August 23-25) in Guangzhou, China,	
4					

A	B
Unnamed: 0	Data
0.0	2021-06-05 00:00:00
1.0	Publication
2.0	Survey publié
3.0	Paul Dorbec (AMACC
4.0	https://link.springer.co
	Data
0.0	2021-06-05 00:00:00
1.0	thèse/HDR
2.0	Nouveau doctorant
3.0	Léo Paviet sera un do
4.0	
	Data
0.0	2021-06-05 00:00:00
1.0	Personnel
2.0	Recrutement d'un anc
3.0	Pablo Rotondo, qui a puis a passé un an (2 a été classé 1er sur u
4.0	
	Data
0.0	2021-06-05 00:00:00
1.0	Publication
2.0	Article
3.0	L'article "Analysis of deux collègues japona
4.0	

AMACC

Aussi une fois que tous les traitements des fichiers xlsx ont été réalisés, voici ce à quoi ressemble un fichier csv sur le serveur. (Un fichier csv qui part pour être archivé)

On remarque que tous les éléments du groupe de recherche sont dans un fichier constitué d'une seule feuille.

On a donc une colonne B remplie de toutes les informations de recherche du mois du groupe de travail AMACC.

IV. Conclusion

1. Réalisé

Ce qui a été réalisé est la principale partie du travail demandé.
Nous avons bien une application client en Python qui demande à l'utilisateur de saisir un fichier xlsx afin de le transformer en page html qui peut être mise sur un site web.
La partie serveur est fonctionnelle, elle permet le traitement des informations, la conversion des données (xlsx > csv) et la création de pages HTML.

Pour ce qui est de la partie Worldpress, le rendu ne pouvant se faire car les fichiers auraient étés trop lourds, cette application est toujours possible. Worldpress fonctionne plus ou moins comme l'émulateur de serveur web que nous avons fait. Il prend un chemin d'accès et affiche le contenu de ce chemin.
Cependant il aurait pu nous apporter une base de donnée importante à la connexion des utilisateurs, et une aide au transfert des fichiers.
Donc l'utilisation du serveur Worldpress ou d'un autre comme Apache2 est toujours envisageable et facilement réalisable.

2. Perspective: améliorations / non réalisé

Comme vu dans le code et ci-dessus, plusieurs points n'ont, délibérément ou par manque de temps, pas été réalisés.

Il est regrettable qu'il n'y ait pas de base de données, ni de communication réseau. Cependant, la réalisation viendrait se greffer facilement à ce qui existe de notre programme. La partie émettrice de l'information serait simplement rattachée au bouton d'envoi de notre programme principal, `UserPage.py`, qui ferait la liste des éléments à envoyer dans le fichier temporaire xlsxTOcsvTemp et enverrais un à un chaque élément.
La partie réceptrice du serveur déclencherait alors la mise en route de la génération des fichiers html ainsi que la conversion en csv.

Les bases de données seraient utiles si les clients (utilisateurs normaux) avaient différents grades ou droits (administrateur).
Nous pourrions aussi ajouter le fait de manager le serveur et la base de données à distance grâce à une interface spécifique aux administrateurs.

On peut voir que plusieurs choses pour des utilisateurs aux droit spéciaux ont été prévues initialement dans le code de l'application, comme le fichier `Dmin.py` qui aurait dû accueillir une page pour administrateur pour générer les rubriques (ce s'est avéré, comme vu plus haut, inutile).
La souplesse du programme à pouvoir switch entre les pages facilement, laisse la porte ouverte à de nouvelles fonctionnalités et même à de nouveaux droits aux utilisateurs.

Nous avions aussi dans l'idée (qui a connu un début de réalisation) de permettre la sélection des rubriques que l'on veut envoyer, au lieu d'envoyer toutes les rubriques.

On peut voir 2 fonctions dans le UserPage.py qui ne servent à rien.
L'une était censée pouvoir permettre la sélection des fichiers à envoyer (select_envoie), et l'autre fonction permettait de créer une liste des informations que l'on envoie.
La liste était donc composée de tous les éléments (fils) qui avaient pour ["texte"] ="envoi".
(Ces éléments sont des structure similaires aux dictionnaires)

En définitive, le code est facilement adaptable et améliorable sans que de grandes modifications soient nécessaires.
Il a été pensé pour avoir plusieurs niveaux de droits, et également pour marcher en client/serveur donc l'ajout de communication est facilement réalisable.