LIGHT UP
YOUR FUTURE_

HARDIS GROUP

reflex
live logistics

Hardis Group dans l'équipe Reflex

Rapport de stage

1er année BTS SIO

A-
On ne voit pas l'aspect demande (problème) et solution (votre réponse à la demande).
Techniquement, le CR est intéressant (le stage doit l'être aussi).
Conclusion correcte (mais ressemble à dernière semaine... ?)

Forest Bastien

Mai – juin 2024

Sommaire :

1. Présentation de l’entreprise

Hardis est une société de services et de conseils en systèmes d'information, son siège social implantée à Seyssinet-Pariset. Créée en 1984, c’est l'un des leaders français du développement et de l'intégration de logiciels. L'entreprise compte aujourd'hui plus de 1300 collaborateurs répartis sur une trentaine d'agences répartit dans le monde. En 2022, elle a réalisé un chiffre d'affaires de 135 millions d'euros.

L'entreprise intervient dans divers secteurs : distribution, industrie, services, administrations. Ses domaines d'expertise couvrent la transformation digitale, les infrastructures cloud et le développement d'applications métiers. Cette polyvalence lui permet d'accompagner ses clients dans leurs projets numériques.

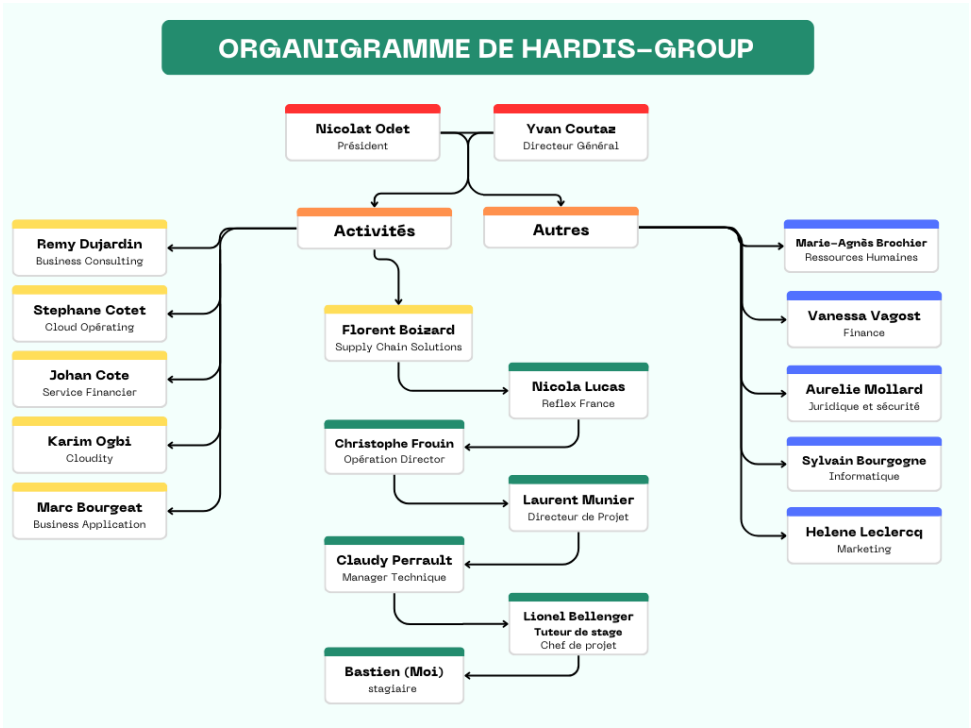
Résolument tournée vers l'innovation, Hardis dispose de centres d'expertise dans le cloud, le big data, la mobilité et les objets connectés. La société consacre chaque année 20% de son chiffre d'affaires à la R&D.

1.1 Situation géographique

Le siège social de l'entreprise est situé à Seyssinet-Pariset, ou je fais mon stage.

1.2 Organigramme de l'entreprise

Voici comment se compose de manière général l'entreprise :



2. Mes missions

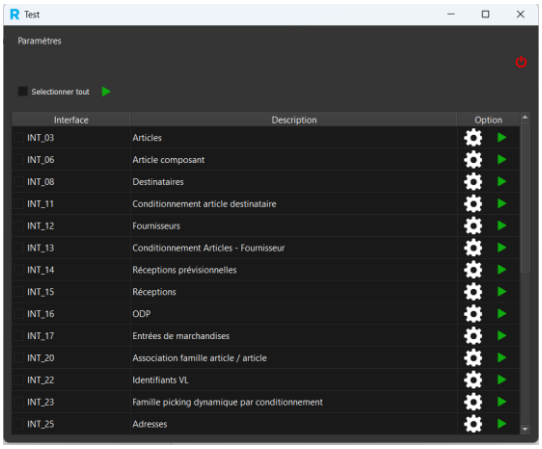
Il y a un nombre d'interfaces déterminé par une base de données, et ces interfaces sont exécutables en ligne de commande, qui extrait de base de données des code activité et dépôts enregistré dans un fichier txt. Le but est de rendre graphiques toutes ces opérations afin de rendre la tâche plus simple. Elle comprend la connexion à une base de données parmi trois types (MSSQL, AS400, ORACLE) qui ont chacun des paramètres de connexion différents et comme on peut le voir dans l'onglet 'Paramètres' il y a 4 autre onglets dans l'application qui permettent de paramétrer l'interface choisie.

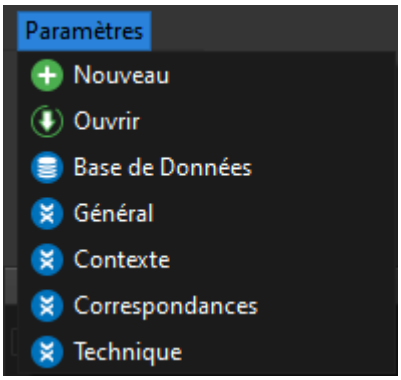
3. Page application

L'application contient plusieurs 'pages' et voici à quoi elles servent

3.1 Interface graphique après la connexion à la base de données :

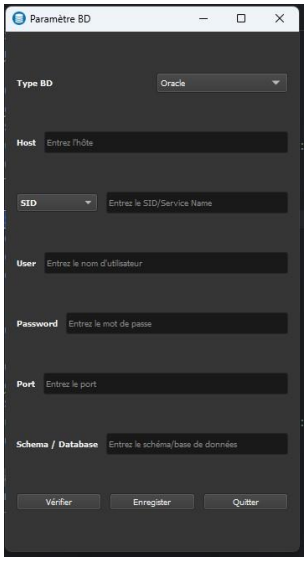
Ici on peut voir une partie des interfaces qui vont être exécuté ainsi que le menue déroulant.

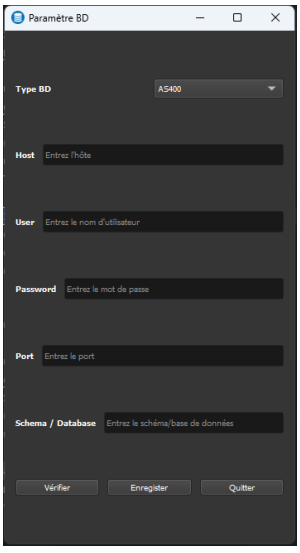


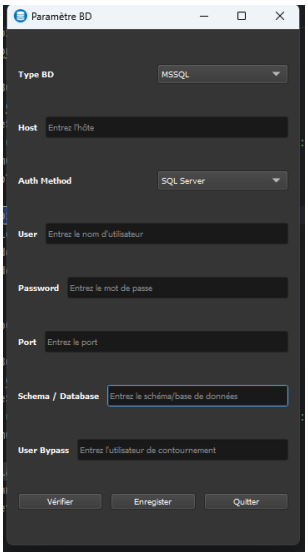


3.2 Connexion à une base de données :

En fonction de la base choisit :

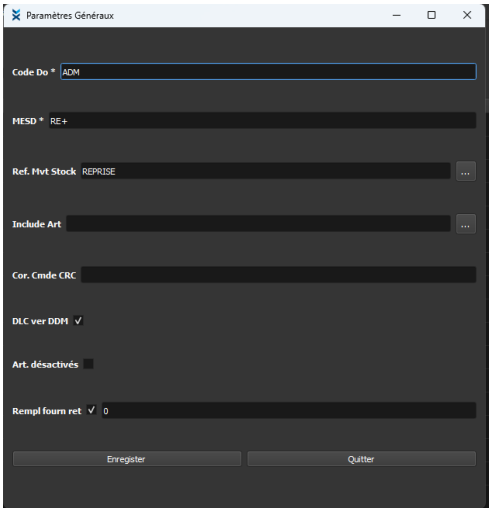






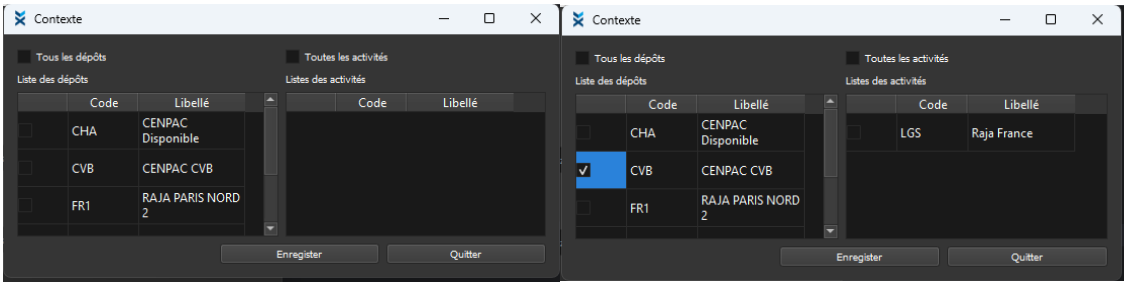
3.3 Page des paramétrages généraux :

Paramètre de base lors de l'envoie de la requêtes SQL pour toute les interfaces



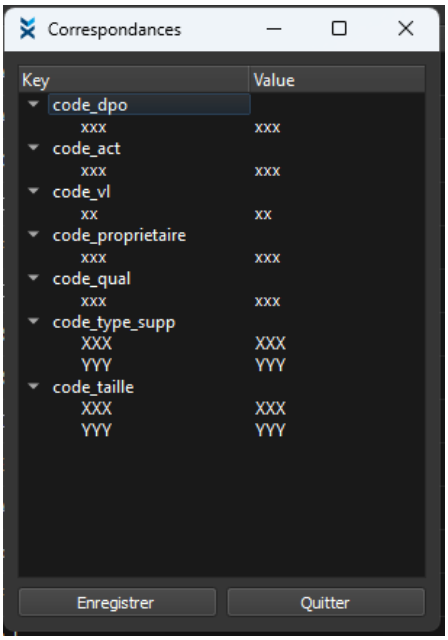
3.4 Page contexte :

Permet de visualiser et d'enregistré les valeurs sélectionnées



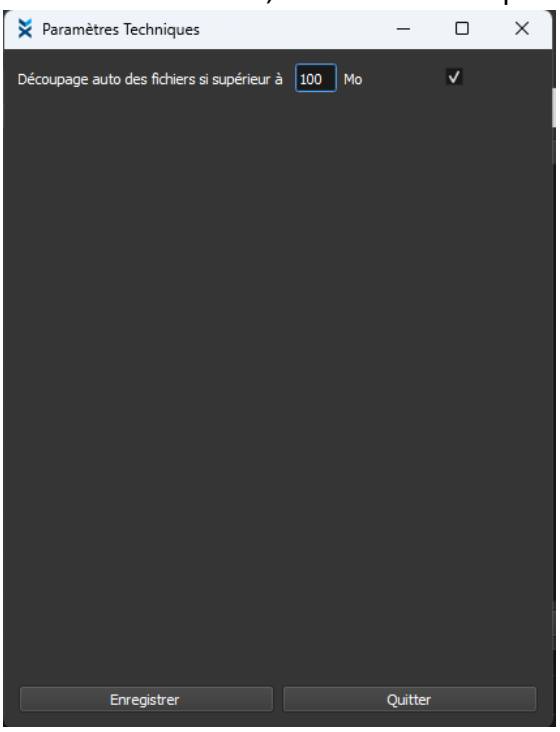
3.5 Page correspondance :

Initialiser via du JSON situé dans une base de données



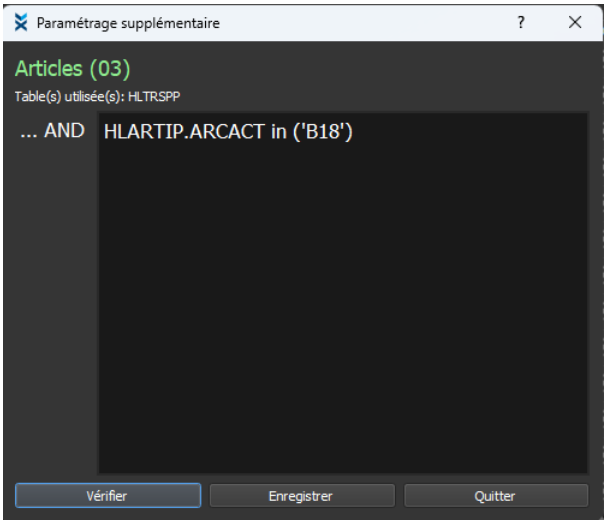
3.6 Page des paramètres techniques :

"Permet le découpage une fois les interfaces lancées si le fichier créé via l'interface est trop volumineux en taille, alors on le découpe via un outil créé par l'entreprise.



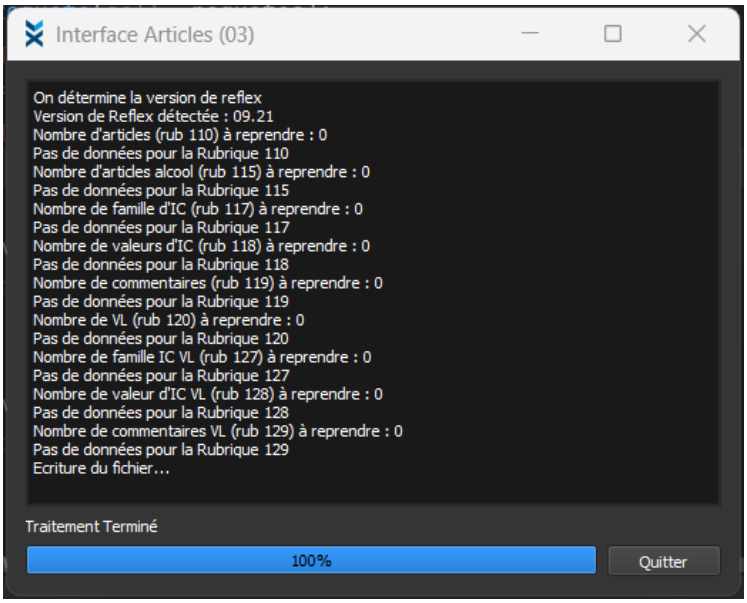
3.7 Pages Paramètre supplémentaire :

Permet d'ajouter du SQL en plus à la requête si l'on veut préciser la recherche.



3.8 Page d’initialisation des interfaces

Cette page permet de voir le suivie de l’avancement de l’opération des interfaces. Le bouton quitter est non cliquable jusqu’à ce que l’opération soit finis comme ici. Le titre et le contenue change en fonction de l’interface choisit.



4. Cette semaine :

Lors de ma troisième semaine, j'ai poursuivi le développement de l'application en cours. Sur la plupart des interfaces (il en manque 2 qui sont spécifiques sur un total de 33), j'ai facilité la communication entre mon application et les interfaces grâce au package Qthread. J'ai ajouté les éléments suivants dans la page d'interface (page affichée au lancement de l'application) :

- Un suivi des requêtes utilisateur (en base de données et dans les journaux (logs))
- L'affichage des journaux générés par l'application (redirection des logs existants)
- Un bouton qui ne s'active que lorsque le téléchargement est terminé ("Quitter")
- Une barre de progression graphique qui suit les téléchargements

4.1 Utilisation de la classe

Cette classe permet de recevoir les différents paramètres de connexion ainsi que les paramètres pour les interfaces. Elle renvoie des informations (requêtes utilisées, barre de progression et suivi de l'avancement).

4.1.1. Les signaux

Ici, on peut voir qu'il y a 4 signaux qui sont utilisés par ces fonctions pour simplifier leur utilisation sur les 33 interfaces.

```
class MyThread(QThread):
    signal = pyqtSignal(str)
    label = pyqtSignal(str)
    ProgBarValue = pyqtSignal(int)
    ProgBarMax = pyqtSignal(int)
    requete = pyqtSignal(list)
```

```
def UpdateValueProgBar(self, value):
    self.ProgBarValue.emit(value)

def UpdateMaxProgBar(self, value):
    self.ProgBarMax.emit(value)

def UpdateLabel(self, value):
    self.label.emit(value)

def contentadd(self, text):
    self.signal.emit(text)
```

Comme le décrivent les noms des fonctions, il suffit de les appeler avec les valeurs qui conviennent, puis d'envoyer le signal.

Programmation de niveau SIO2... cool !

4.1.2. La réception du signale

La partie encadrée en rouge permet d'importer l'interface sélectionnée et la ligne en dessous permet de l'appeler avec les paramètres de connexion et les paramètres choisis en vert. La partie en bleu permet de recevoir le signal et de le rediriger vers une fonction. Et la partie encadrée en jaune permet de rendre le bouton cliquable car il est désactivé pendant toute l'opération.

```
def run_interface(self):
    corr = Correspondance(self)
    correspdata = corr.CorrespJson(self.curTb)
    module_name = self.intferfselec
    module = __import__('Int', fromlist=[module_name])
    interface_module = getattr(module, module_name)
    self.thread = interface_module.MyThread(self.params, correspdata, self)
    self.thread.signal.connect(self.update_text)
    self.thread.label.connect(self.update_label)
    self.thread.ProgBarValue.connect(self.UpdateProgBarValue)
    self.thread.ProgBarMax.connect(self.UpdateProgBarMax)
    self.thread.requete.connect(self.SendRequete)
    self.thread.start()
    self.thread.finished.connect(self.finish)
```

4.2 Requêtes utilisées

En cas de problème ou de plantage de l'application, il faut pouvoir avoir un suivi de l'interface utilisateur dans les journaux. Cela était déjà mis en place à mon arrivée, mais on m'a demandé d'enregistrer les requêtes utilisateur en base de données et dans les logs.

4.3.1. Partie Interface

La fonction WriteRqt est appelée à la fin de chaque interface. Elle permet d'ajouter les requêtes utilisées après les logs (qui n'avait pas été fait par moi).

```
def WriteRqt(self):
    logger.info("**************** Requete utilise ********************")
    for requete in self.TabRqt:
        logger.info(requete)
    logger.info("***************** FIN ********************")
    self.requete.emit(self.TabRqt)
```

4.3.2. Partie application

Lorsque la fonction WriteRqt est appelée, on reçoit un signal avec un tableau en paramètre qui contient toutes les requêtes utilisées à insérer dans une base de données, dans un champ de type BLOB (qui ne comporte pas de limite de caractères sur SQLite).

```
1 usage  new *
def SendRequete(self, requetes):
    currqt = self.connTb.cursor()
    sql = 'INSERT INTO requete (rqt) VALUES (?)'
    currqt.execute(sql, (self.intferfselec, requetes))
    self.connTb.commit()
```

5. Conclusion

Ce stage dans le développement informatique chez Hardis Group a été une expérience enrichissante qui m'a permis d'approfondir mes connaissances en Python, notamment sur les paquetages pour les bases de données et la communication entre plusieurs classes de façon dynamique. C'est une expérience très positive pour ma part.

6. Tableau Ressenti

Lundi	Mardi	Mercredi	Jeudi	Vendredi