



Glory Global Solutions

Rapport de Stage

Réalisé par

Guillot Natthan

natthan_guillot@etu.u-bourgogne.fr

Supervisé par

Picovschi Matthieu

matthieu.picovschi@fr.glory-global.com

1 juillet, 2024

SOMMAIRE //

Table des matières

SOMMAIRE //	2
Introduction/	4
Informations sur l'entreprise/	4
a) Histoire.	4
b) Concurrence.	5
c) Chiffres clés.	5
d) Produits.	5
e) Clientèle.	6
f) Collaborateurs.	7
g) Organigramme.	7
Ma Mission/	8
a) Robot Disjoncteur.	9
• Envoyer un message	10
• Recevoir une entrée	10
• Partie Electronique	11
• Code Arduino.	13
b) Ecriture/Analyse des transactions.	14
• TransactionalReaderWriter.cs	15
• SecureReaderWriter.cs	16
• Autres Fonctionnalités	17
• Connection Client/Serveur	18
c) Automatisation.	19
Analyse des Résultats/	21
d) HHD/Win7.	21

• 1/100ko, 100 dossiers – WriteThrough,	21
• 1/100ko, 200 dossiers – WriteThrough	22
• 1/100ko, 200 dossiers – None	22
• 1/100ko, 200 dossiers – Asynchronous	22
• 10Mo, 100 dossiers – WriteThrough,	22
• 50Mo, 50 dossiers – WriteThrough,	23
• 100Mo, 25 dossiers – WriteThough,	23
e) SSD/Win10.....	24
• Fausses méthodes d'écritures	24
• 1/100ko, 200 dossiers – No WriteThrough/WriteThrough	24
• 10Mo, 100 dossiers – No WriteThrough/WriteThrough	24
• 50Mo, 50 dossiers – No WriteThrough/WriteThrough	25
• 100Mo, 25 dossiers – No WriteThrough/WriteThrough	25
• Conclusion des résultats.....	25
Retours sur le stage/.....	27
Expérience humaine.	27
Difficultés rencontrées.....	27
Conclusion/.....	28

Introduction/

Durant cette période de stage qui se tiendra du 1er au 26 Juillet 2024, j'ai pu me faire intégrer au sein de l'entreprise Glory Global Solutions. C'est donc pendant cette période que je devrais mettre mes compétences acquises au cours des années précédentes afin de trouver ma place dans ce milieu professionnel. Il en va de soi que si je suis ici c'est aussi et principalement pour apprendre, il faudra donc que je me montre curieux et investit afin de pouvoir aider l'équipe à réaliser les tâches qui me seront confiées.

C'est pendant ce stage surtout que nous pouvons nous apercevoir de ce qui est attendu en tant qu'ingénieur au sein d'une entreprise. Une question sera aussi importante à soulever : Quelles sont les différences entre les enseignements qui nous sont fournis à l'école et ce que l'on apprend réellement sur le terrain ?

Informations sur l'entreprise/

Glory Global Solutions est une entreprise spécialisée dans les technologies liées à l'argent liquide. Concrètement, leur gamme de produits/services et leur secteur d'activité est très large. De la machine de gestion automatique du liquide dans les supermarchés jusqu'aux logiciels sur les automates bancaires en passant par l'entretien et le support sur ces dernières, Glory sait se positionner en force à l'international pour dominer la gestion d'espèce. Présentons donc son activité en plusieurs petits points :

a) Histoire.

Il nous faut remonter jusqu'en 1918 lors de la création de « Kokuei Machinery Manufacturing » au Japon dans la ville d'Himeji. A l'origine, cette entreprise fut fondée dans le but unique d'assurer la création et la réparation d'ampoules. Puis l'année suivante les voilà entrain de renforcer leur technologie afin d'en arriver à la fabrication de la première machine à compter les pièces. S'en suit plusieurs années de développement jusqu'en 1950 où Kokuei décida de s'investir à plein temps dans le business des machines gérant les espèces. Avec de grandes ambitions telles que devenir pionnier dans ce domaine de la gestion d'argent, il sort en 1958 des machines de ventes automatique pour les chewing-gums et les cigarettes. C'est la première au Japon, ce qui augmente considérablement la renommée de Kokuei. Il faudra attendre 1971 pour voir le nom de Glory apparaître. Glory développe alors deux technologies importantes : « reconnaissance » et « identification » elles sont la clé de la rapidité avec laquelle les machines traitent l'argent. C'est en améliorant ces traits sur les différentes années que l'on a eu « l'open Teller system » en 1986 et le Recycleur d'espèces pour caissiers. Deux outils qui existent encore et qui ont permis à Glory d'avoir une clientèle telle que les banques, les supermarchés ou bien les simples magasins.

b) Concurrence.

Malgré l'avantage qu'ils ont en termes de technologie, d'autres entreprises essayent tout de même de se hisser au niveau de Glory Global Solutions afin de leur créer de la concurrence. Pour les empêcher de prendre le monopole mondial dans ce domaine de la gestion d'espèce. Parmi ces entreprises, on comptera les quelques principaux tels que Wittenbach situé en Suisse, STL situé en Angleterre et pour finir BelCash situé en Ethiopie. Cependant, Glory reste bien mieux implantée dans le marché du fait qu'il se développe à l'international.

c) Chiffres clés.

Date de fondation	1918
Capital	74 675 952,50 €
Nombre d'actions	58 938 210
Nombre d'employés	11 398
Nombre d'entreprises	102
Nombre de pays concernés	37

d) Produits.

Comme on a pu le voir précédemment, Glory est spécialisée dans les automates qui servent dans le milieu bancaire principalement. On retrouve donc quelques modèles emblématiques :

- **Caisses automatiques à recyclage (GLR-200)** : Son but est de faciliter le paiement en espèce afin que le personnel puisse entièrement se consacrer à la partie interaction humaine, laissant ainsi la machine traiter toute l'espèce que le client lui donne.



- **Solutions de dépôt (GDB-100)** : Permet de trier les billets qui lui sont remis très rapidement, ce qui peut être très pratique pour les entreprises. Il est généralement utilisé pour ranger les billets à disposer ensuite dans les DABs.

- **Compteuses de billets (GFB-800)** : Comme son nom le précise, cette machine est très utilisée dans les commerces afin de compter aisément et rapidement les billets. Ce modèle précisément permet aussi de détecter les contrefaçons.



- **Trieuses de billets (UW-F)** : La gamme UW-F est très similaire aux solutions de dépôts, mais ne permet pas cette fonctionnalité. Elle est utilisée pour sa très grande capacité. Sa vitesse reste tout aussi impressionnante : 1000 billets par minute.

- **Solutions pour pièces (SCW-20)** : Comme l'on peut s'en douter, trier, voire compter les pièces peut-être une tâche très fastidieuse. C'est pour cela que ces machines ont été créées. Elles permettent de compter nos pièces et de les trier. Tout ça en une fraction de seconde à la vitesse de 2000 à 3000 pièces par minute.



- **Libre-Service (COMBO Kx06)** : Cette série d'automates est spécialement prévue pour le 2 en 1. Utilisées dans les agences bancaires, ces machines sont utiles dans le cas d'un dépôt, ou même celui d'un retrait.

- **Points de vente (CI-10X)** : Permet de traiter automatiquement les transactions d'espèces dans les magasins de vente au détail. Ce qui minimise la manipulation de monnaie et donc augmente la sécurité.



e) Clientèle.

Glory s'est rependue dans de nombreux domaines à travers le monde. On parle ici de vente au détail avec ce qui se passe chez les commerçants, dans lesquels ils s'occupent de la gestion d'espèce automatisée jusqu'à la remise en banque de l'argent. De ce fait, Glory a aussi des clients dans les banques centrales et les institutions financières où leurs machines permettent plus rapidement de traiter l'importante quantité d'argent qui s'y trouve. Pour finir, L'entreprise se positionne aussi dans les Casinos comme dans les restaurants pour faciliter et sécuriser les transactions financières qui s'y déroulent.

f) Collaborateurs.

Afin de faire évoluer une entreprise et la faire perdurer, il est important d'avoir des collaborateurs. De ce fait nous allons voir un par un les plus connu travaillant avec Glory Global Solutions.

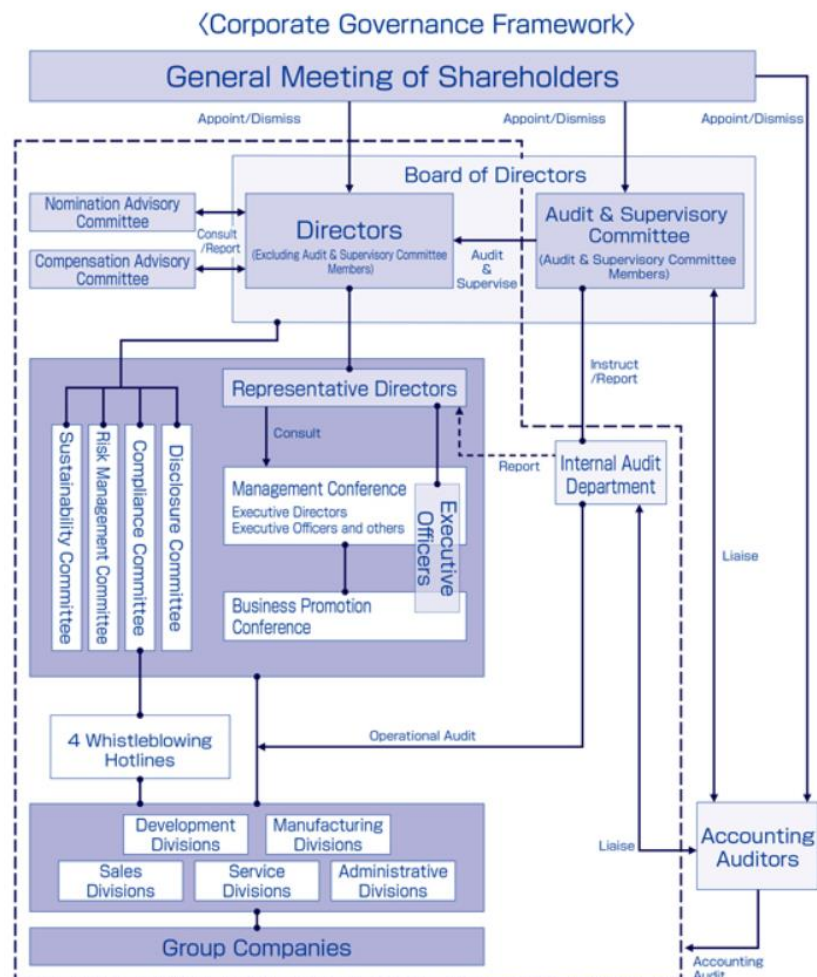
Premièrement, nous avons la **banque centrale européenne** qui est-elle même cliente mais en tant que partenaire, Glory peut donc afficher en temps réel des informations sur les émissions de billets de banque sur toute la zone Euro.

VERINT, leader mondial des solutions d'optimisation de la force de travail en entreprise permet aux entreprises de Glory de bénéficier d'une amélioration de la qualité de leur service client tout en augmentant pas les coûts liés avec le personnel.

FISERV, quant à elle est une entreprise réalisant des logiciels gérant la logistique des espèces. L'association de la connaissance de Glory quant à la création de ces machines et Fiserv pour le software permet la création de ces machines permettant de trier efficacement la monnaie.

g) Organigramme.

Glory global solutions étant une entreprise mondialement connue, il est difficile de faire l'organigramme en entier. Voici donc les personnes importantes ainsi que quelques personnes de la succursale française à Croissy.



Ce schéma nous montre un peu comment l'entreprise fonctionne. Voici donc quelques personnalités importantes qui permettent une telle entreprise de fonctionner :

1. **Toshimitsu Yoshinari** : Président-directeur général et Président du Conseil d'Administration. Toshi a rejoint Glory en 1998 et a occupé divers postes, notamment Chief Solutions Officer. Il possède une vaste expérience internationale.
2. **Michael Williams** : Directeur financier et directeur des opérations. Michael est en charge des finances, de la chaîne d'approvisionnement et de la transformation des processus chez Glory. Son expertise en finance a contribué à la croissance et à la rentabilité de l'entreprise.
3. **Tomoko Fujita** : Directrice du développement d'entreprise et de la planification. Tomoko supervise la planification stratégique, les ressources humaines et le marketing pour la division International Business. Elle joue également un rôle clé dans les fusions et acquisitions.
4. **Takuya Onoe** : Directeur général exécutif, Core Solution Office. Takuya est responsable des ventes centrales aux clients des secteurs de la finance, de la vente au détail et de l'alimentaire.

Ma Mission/

Durant ce stage d'une durée d'un mois, il m'a été confié une mission nécessitant à réaliser des tests sur un ordinateur XE2 qui est généralement le cerveau des automates bancaires. En temps normal, les ATM utilisent un logiciel s'appelant « atmIp » permettant à ces derniers de gérer les transactions. Quand une transaction est réalisée, un dossier est créé avec la date de cette dernière. On a créé ensuite trois fichiers, le journal, l'historique (s'il y a plusieurs transactions) et les informations. Quand tous ces fichiers ont été créés, un autre fichier fait son apparition : le « commit.txt » si son contenu et sa création ont bien été faits, alors la transaction est considérée comme valide. Les transactions sont ainsi envoyées vers une sorte de cloud toutes les cinq secondes, ce qui permet de ne pas perdre de données.

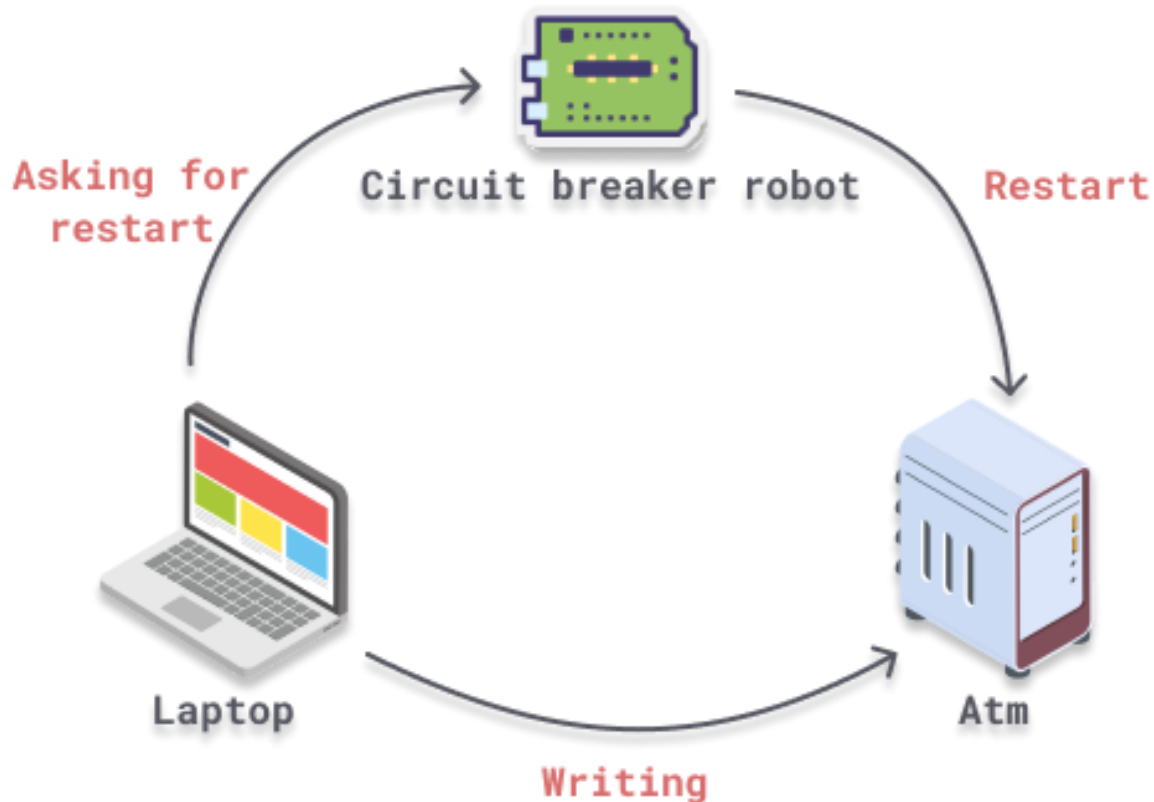
Une problématique se pose donc : lors de l'arrivée des transporteurs de fonds, la plupart se permettent autant par gain de temps que de simplicité de couper le courant à la place d'attendre qu'il s'éteigne de lui-même. Nous voulons alors vérifier si ces coupures sont la cause de données corrompues et comment donc régler voire atténuer ce souci.

Il a donc fallu trouver un moyen de construire un modèle afin de reproduire les coupures de courant pour analyser les conséquences de ces dernières. On peut donc dispatcher ma mission au sein de Glory en trois points importants :

- **Robot disjoncteur** (réaliser les coupures de courant).
- **Programme Ecriture/Analyse** (permet l'écriture des transactions puis les analysent)
- **Robot Framework** (utile pour automatiser des scénarios)
- Analyse des résultats sur Excel (Solutions)

a) Robot Disjoncteur.

Le projet du robot disjoncteur a un objectif très simple, il fait partie du projet dans le seul et unique but de pouvoir contrôler efficacement les coupures de courant. De ce fait on utilisera simplement un une Arduino ainsi qu'un ordinateur qui lui communiquera l'information de redémarrer. Voici un schéma qui représente en général ce que nous allons devoir réaliser pour ce projet :



Maintenant que nous pouvons voir comment cela va pouvoir se profiler, il ne reste qu'une seule chose à faire : Analyser ce qu'il se passe au niveau de notre Arduino, est plus précisément sur notre Robot Disjoncteur.

Pour gérer le fait de permettre à un ordinateur de redémarrer via un Arduino est une chose plutôt compliquée. Elle nécessite de comprendre comment ce microcontrôleur fonctionne pour communiquer afin de pouvoir interagir avec.

Deux choses sont donc à faire :

- Communiquer l'action pour faire redémarrer
- Couper le courant de l'ordinateur cible sans pour autant couper le courant de l'Arduino

De ce fait, commençons par comprendre comment émettre des informations vers l'Arduino. Le microcontrôleur que nous utilisons possède une librairie intégrée qui utilise le port série de ce dernier afin de lui permettre de communiquer autant en entrée qu'en sortie. De ce fait c'est le plan parfait si l'on veut afficher des informations dans la console ou bien lui en transmettre par ce même biais.

• Envoyer un message

Comme vous pouvez le voir ci-dessous, nous avons ici un code en Arduino permettant d'écrire le texte de base « Hello World » sur le port série. Ce qui signifie que toute personne connectée sur ce même port au moment de l'envoi verra le message s'afficher. A noter qu'il est primordial d'initialiser notre port série au début, sinon rien ne sera pris en compte.

```
// Little Arduino code to write hello world to a serial monitor
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

void loop() {
  // print the string "Hello World" to the serial monitor
  Serial.println("Hello World");
}
```

• Recevoir une entrée

Aussi appelées « inputs » en anglais, ces méthodes sont très pratiques puisqu'elles permettent à notre utilisateur d'interagir avec le programme. De ce fait, on met le code en attente jusqu'à ce que notre utilisateur ait rentré un mot clé choisi au préalable puis selon le code on agit en conséquence. Voici comment modéliser ça pour mieux comprendre.

```
// Little Arduino code to manage inputs
void setup()
{
  // Initialize serial communication at 9600 bits per second
  Serial.begin(9600);
}

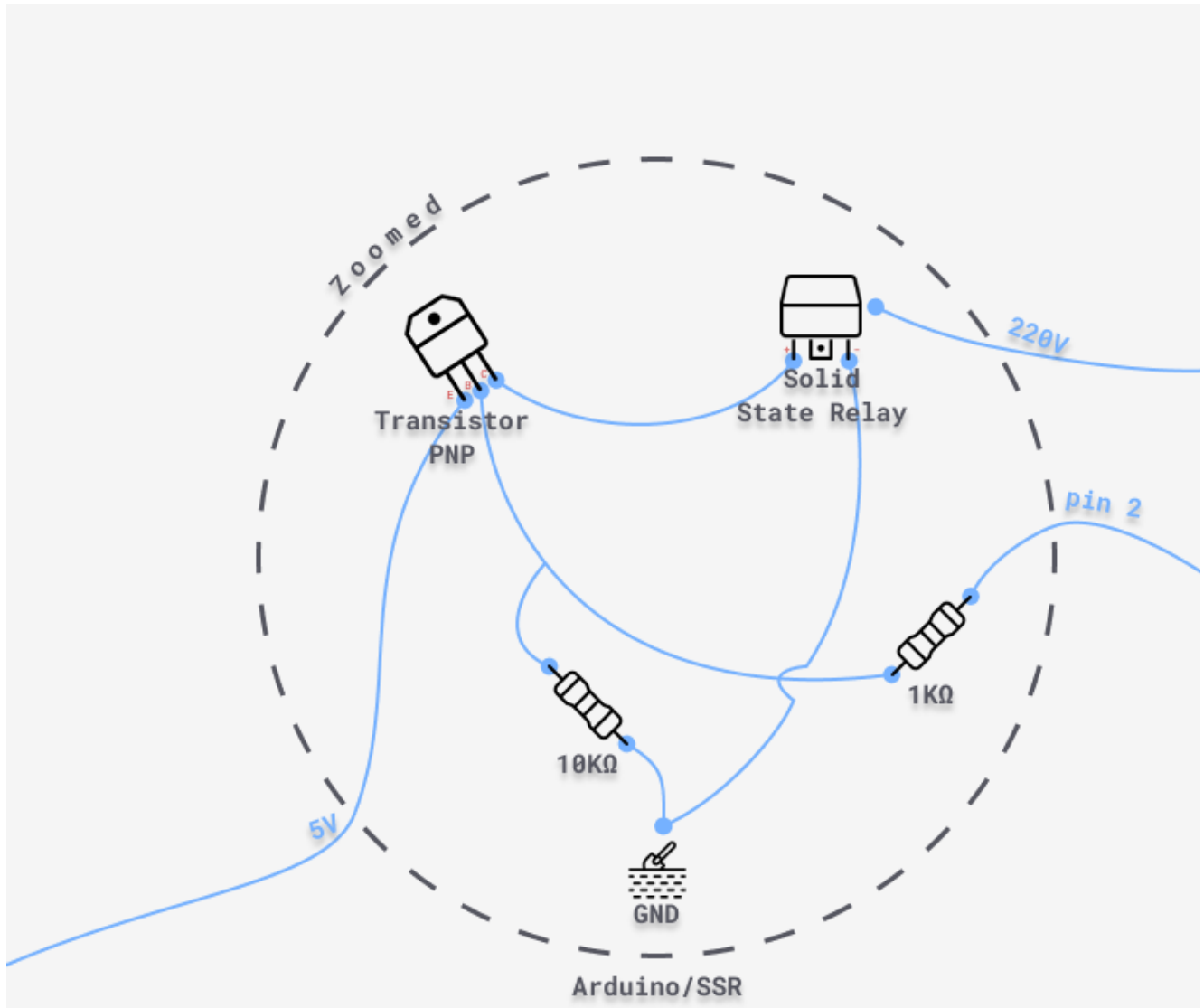
void loop()
{
  // Check if the Serial port is available
  if (Serial.available() > 0)
  {
    // Read the incoming string
    String command = Serial.readStringUntil('\n');

    // Remove any trailing newline characters from the command
    command.trim();

    // Execute commands
    if (command == "hello")
    {
      Serial.println("Hello, World!");
    }
    else
    {
      // If an unknown command is received, notify the user
      Serial.println("Unknown command");
    }
  }
}
```

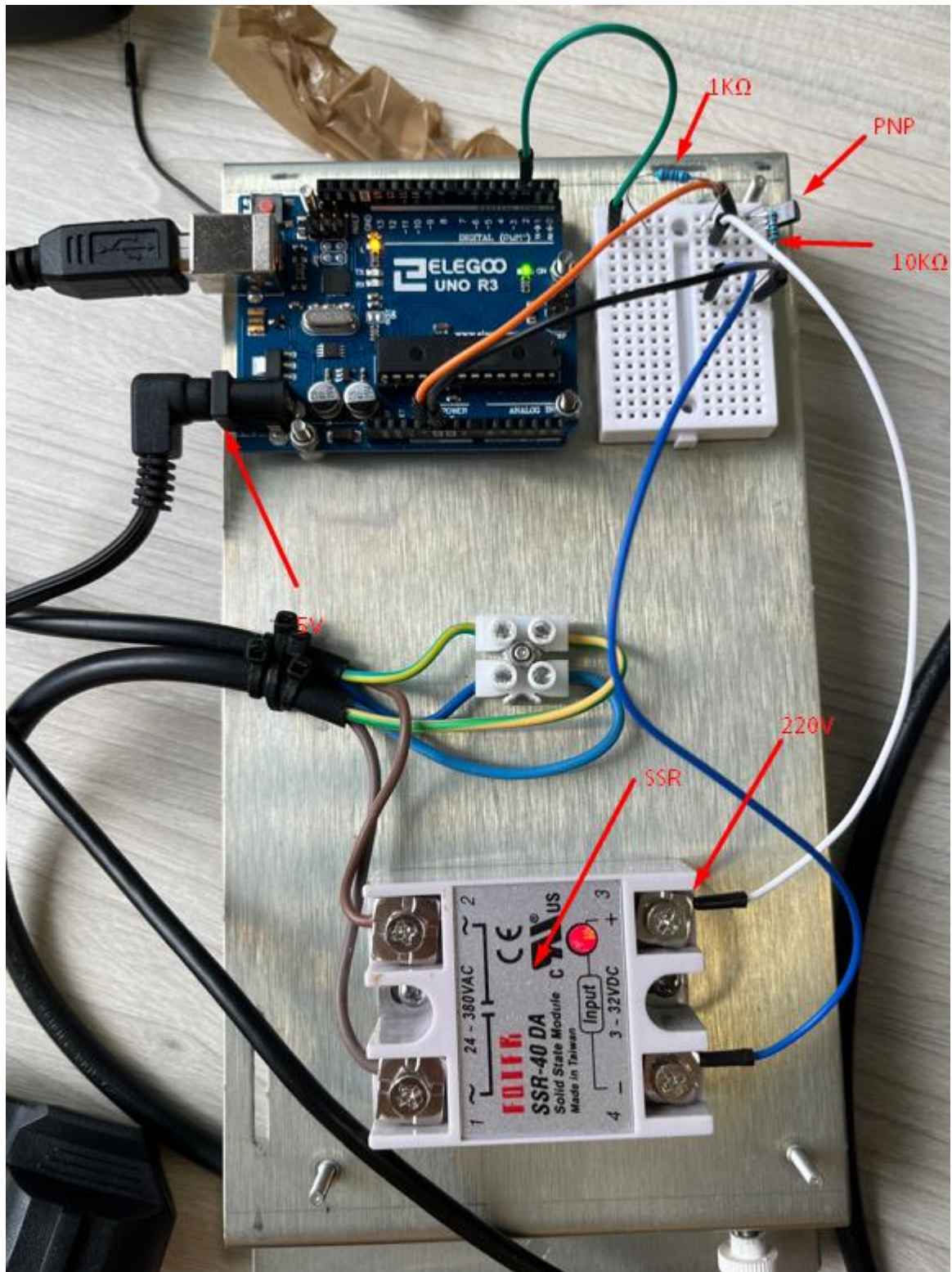
- **Partie Electronique**

Maintenant que nous savons comment traiter le fait de recevoir ou d'envoyer des données, nous allons maintenant nous attaquer à la partie électronique : la partie qui s'occupera d'éteindre l'ordinateur. Avant de commencer, faisons comme précédemment le schéma électronique de notre montage :



Examinons un peu ce dernier et voyons ce qu'il fait. Il faut d'abord savoir que le transistor PNP que nous utilisons a une tension de seuil d'environ 0,7V. C'est-à-dire que si la tension au niveau de la base B est supérieure à cette valeur, alors le transistor peut alors conduire le courant entre l'émetteur E et le Collecteur C. De ce fait, le Solid State Relay est alors alimenté, ce qui alimente notre ordinateur en 220V et inversement si la tension sur B est inférieure au seuil. Un problème nous était remonté : quand on se connectait au port série pour y transmettre des données, le pin 2 arrêta de transmettre vers le SSR, ce qui coupait aussi l'ordinateur. Il a donc fallu trouver une solution : ce transistor avec la résistance de 10KΩ nous assure que quand même s'il y a des incertitudes ou du bruit, cela ne désactive pas SSR.

Si l'on veut voir à quoi cela peut-il ressembler en vrai, voici une petite photo pour s'en donner une idée :



De ce fait on peut voir que le relai est assez imposant, de plus, il y a cette LED très utile nous permettant de voir si oui ou non le SSR transmet du 220V. Le cas de la LED allumée nous permet de voir que l'ordinateur est bien sous tension.

• Code Arduino

Enfin, pour finir sur cette partie de ce Robot permettant de gérer si oui ou non on délivre 220V à l'ordinateur cible, il ne nous reste plus qu'à présenter ce que fait le code Arduino de ce dernier :

```
#include <Arduino.h>

// Pin number for the SSR
const int ledPin = 2;

// Method to turn the LED on (enabling the SSR)
void turnLedOn()
{
    digitalWrite(ledPin, LOW);
}

// Method to turn the LED off (disabling the SSR)
void turnLedOff()
{
    digitalWrite(ledPin, HIGH);
}

// Method to restart (power off, wait for 10 seconds, and power on)
void restart()
{
    Serial.println("Restarting...");
    delay(1000);
    turnLedOff();
    delay(10000);
    turnLedOn();
}

void setup()
{
    // Initialize the digital pin as an output.
    pinMode(ledPin, OUTPUT);
    // Initialize serial communication at 9600 bits per second
    Serial.begin(9600);

    // Turn on the LED at the start of the day
    turnLedOn();
    delay(1000);
}

void loop()
{
    // Check if data is available to read
    if (Serial.available() > 0)
    {
        // Read the incoming string
        String command = Serial.readStringUntil('\n');

        // Remove any trailing newline characters from the command
        command.trim();

        // Execute commands
        if (command == "o")
        {
            turnLedOn();
        }
    }
}
```

```

else if (command == "f")
{
    turnLedOff();
}
else if (command == "RESTART")
{
    restart();
}
else
{
    // If an unknown command is received, notify the user
    Serial.println("Unknown command");
}
}
}

```

Les seules choses qui changent avec ce que l'on aurait pu voir précédemment, c'est que nous allons utiliser `digitalWrite` qui permet de gérer la tension sur les pins « digitaux ». De ce fait quand on met le SSR (pin2) sur « LOW » (pas de courant ne sort), ce qui signifie que le transistor sera bloquant et donc l'ordinateur cible aussi. Puis inversement si nous mettons le pin2 sur « HIGH ». Nous voilà donc avec notre robot coupeur de courant qui fonctionne et qui n'attend maintenant que d'être commandé !

b) Ecriture/Analyse des transactions.

Maintenant que nous avons la possibilité de simuler nos coupures de courant, il est temps de voir comment nous pouvons écrire nos transactions sur l'ordinateur. Etant donné la durée du stage, et pour des soucis d'homogénéité, j'ai pu reprendre le projet du nom « d'atmlp » qui s'occupe justement de cette écriture, mais aussi de leur enregistrement et de la gestion des automates. Cependant, du fait que ce soit un projet imposant, je ne pouvais pas me permettre de récupérer l'ensemble de ce dernier afin de travailler dessus. Il a donc fallu trouver les fichiers qui m'étaient utiles pour écrire des fichiers ou bien les lire. Ils sont au nombre de deux :

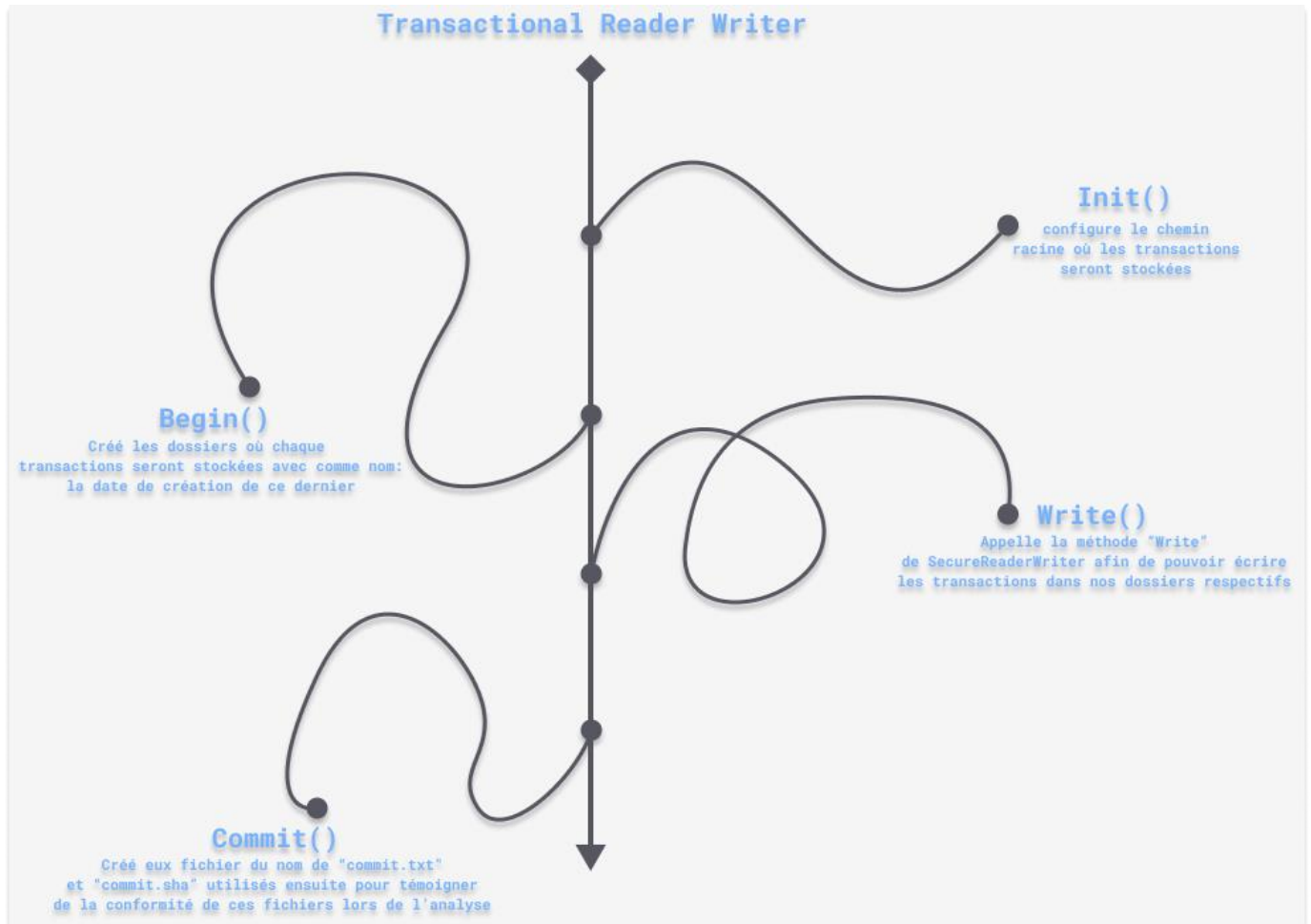
- **TransactionalReaderWriter** qui est un code de haut niveau qui appellera
- **SecureReaderWriter** un code qui est donc de plus bas niveau

Le but de cette manœuvre est donc d'avoir un code que l'on peut aisément appeler depuis le « main » qui appellera par la suite le code de plus bas niveau. Ceci permet de structurer correctement son code en plus d'en gérer la sécurité.

L'une des choses les plus complexes au début c'est bien de se mettre dans le bain et comprendre le code d'un projet auquel on n'a jamais touché auparavant. Etant donné que le nom des fonctions était implicite et le code structuré, il était donc plus simple d'opérer. Si j'ai bien appris une chose sur cette partie c'est que la notion de structurer son code pour le rendre lisible aux yeux des autres est donc très importante pour qu'il puisse être repris par la suite. Il va donc maintenant falloir regarder ce qui se passe à l'intérieur de ces codes dans les grandes lignes, à quoi servent-ils, comment ils fonctionnent et comment les utiliser ? Trois questions auxquelles nous allons répondre dès à présent.

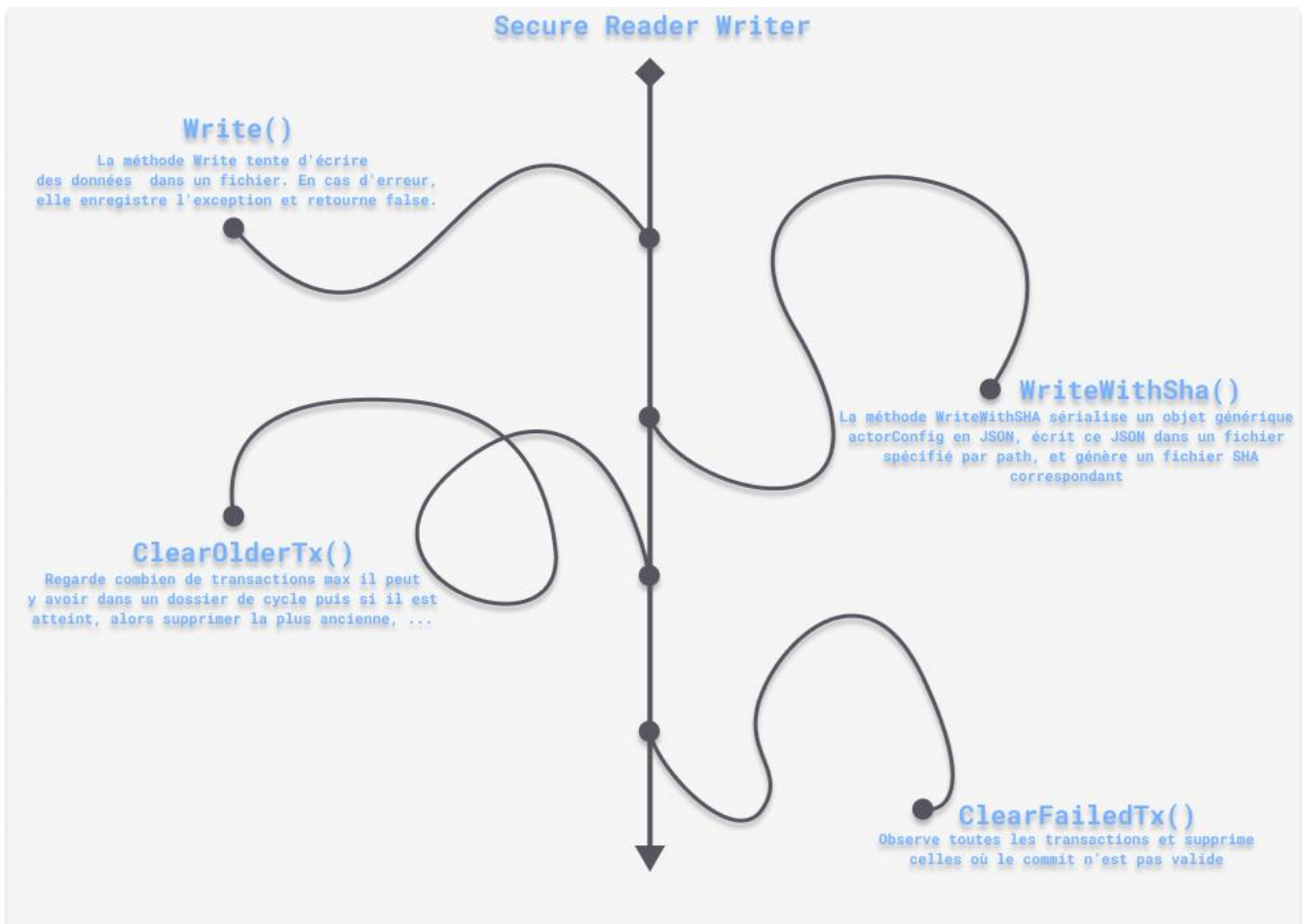
- **TransactionalReaderWriter.cs**

Avant de se poser une seconde pour commencer à analyser le code, voyons tout d'abord comment il fonctionne dans les grandes lignes. Il sera ensuite plus simple de voir ce que retourne chaque fonction.

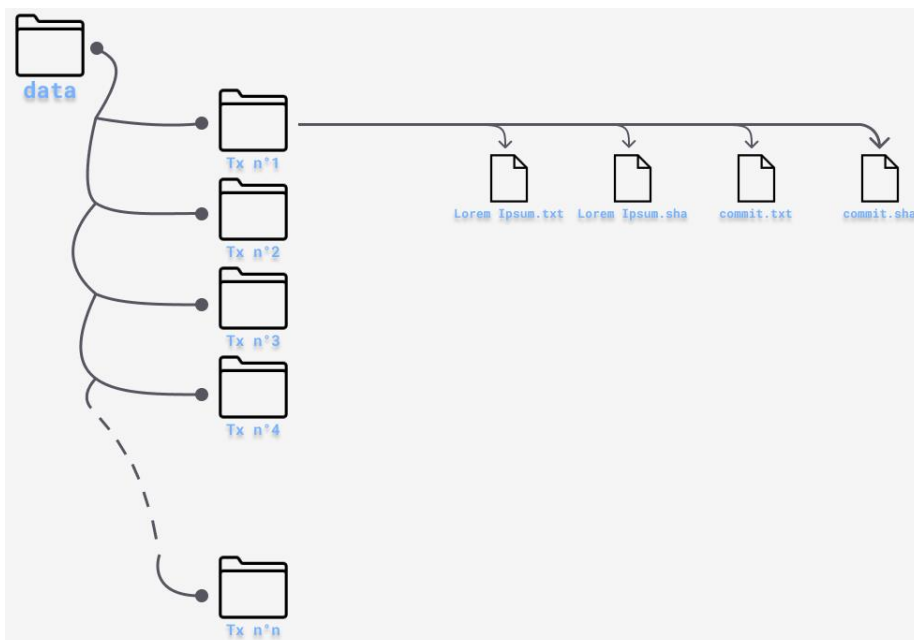


Il reste tout de même à noter que les fichiers sont en C# qui est un langage ressemblant grandement au C++ avec quelques petites subtilités.

- SecureReaderWriter.cs



Voici donc comment ces deux fonctions fonctionnent. Cependant, ce n'est pas tout, car le but de notre programme c'est d'écrire des fichiers puis de les analyser. Dans le cas de notre écriture, on se retrouve à créer un dossier « cycle » qui contiendra nos transactions. Chaque dossier de transactions quant à lui contiendra 4 fichiers : un « **Lorem Ipsum** » pour simuler le contenu de la transaction et ainsi pouvoir en faire varier la taille et le commit pour valider la fin d'une transaction. Ces deux fichiers auront tous deux leur « **.sha** » afin de permettre une vérification supplémentaire. Voici à quoi peut donc ressembler notre hiérarchie de fichier :



• Autres Fonctionnalités

Dans le code que je mettrai à disposition à la fin de ce rapport, nous avons eu besoin d'utiliser d'autres fonctions pour que tout fonctionne à la perfection. Il a fallu par exemple générer les Lorem Ipsum, analyser les résultats, les exporter sur Excel, Récupérer les paramètres passés en ligne de commande ou bien même utiliser une librairie Kernel32 pour utiliser une autre manière d'écrire au plus bas niveau. Le but étant de plus tard changer la taille des fichiers écrits afin de savoir si cela a un impact sur la corruption des fichiers, le Lorem Ipsum est un modèle parfait puisqu'il permet de nous générer des fichiers de taille précise sans trop s'embêter. Maintenant, pour que l'ordinateur analyse nos données, il nous fallait dans un premier temps discuter de ce qu'il était important de relever.

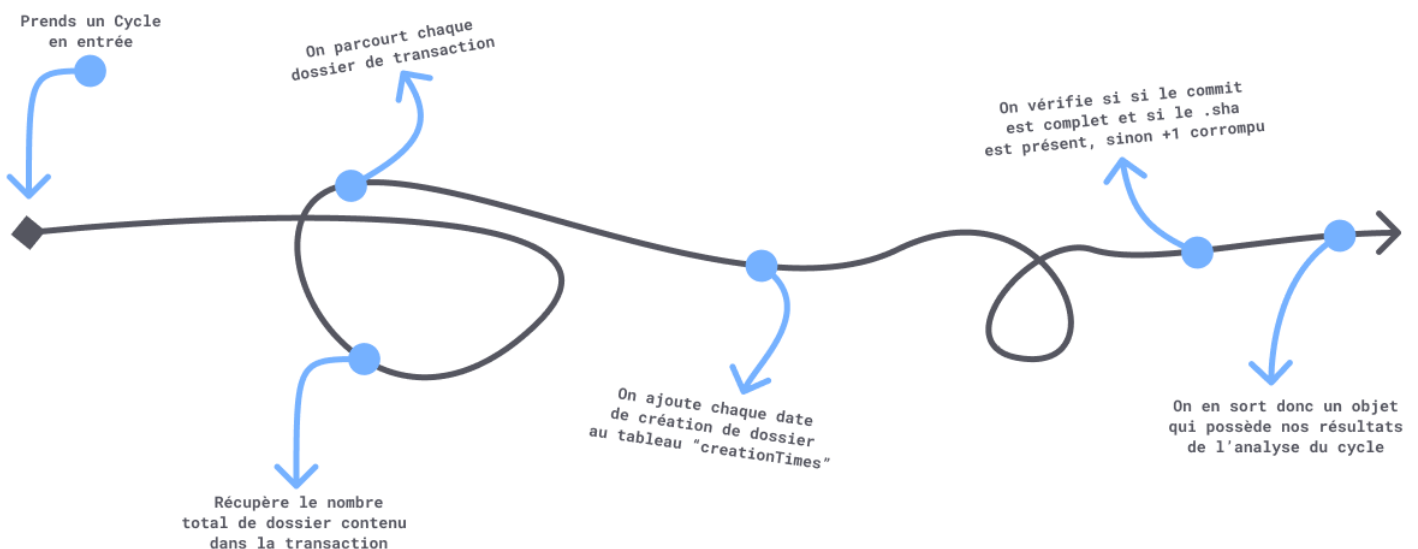
Après avoir discuté, nous en avons déduit que ces informations suivantes étaient pertinentes pour l'analyse:

- Le nom du cycle
- Le nombre total de transaction dans le cycle
- Un tableau contenant la date de création de tous les dossiers
- Le nombre de transactions corrompus

De ce fait voici comment l'analyse de chaque cycle se réalise :

De ce fait on se retrouve à la fin avec tous nos résultats prêts à être utilisés pour notre Excel.

CycleAnalysisResult.cs



En parlant de cet outil capable de réaliser des tableurs, j'ai pu apprendre à utiliser une librairie C# du nom de « **ClosedXML.Excel** ». Comme il est possible de le voir sur le GitHub dans le fichier « ExportExcel.cs », Nous allons créer un fichier Excel ou l'ouvrir s'il est déjà présent, pour ensuite créer différentes feuilles avec le nom de la méthode d'écriture ainsi que le type de stockage utilisé pour ainsi dans chaque feuille remplir une ligne par cycle avec toutes nos informations récupérées. Une chose est faite en plus : on calcule le temps min, max et moyen qu'il faut pour écrire une transaction.

Numéro de Cycle	Nombre de dossier	Nombre de corrompus	Taille d'une Tx	Temps min	Temps max	Temps moy
1	201	11	1000o	3529.65ms	...	
2	...					
3						
4						
5						
...						

Voici donc ce qui est obtenu comme valeur lors de l'exportation en fichier Excel. Nous le verrons dans la prochaine grande partie, nous verrons en quoi toutes ces valeurs jouent un rôle important dans l'analyse de nos données par la suite.

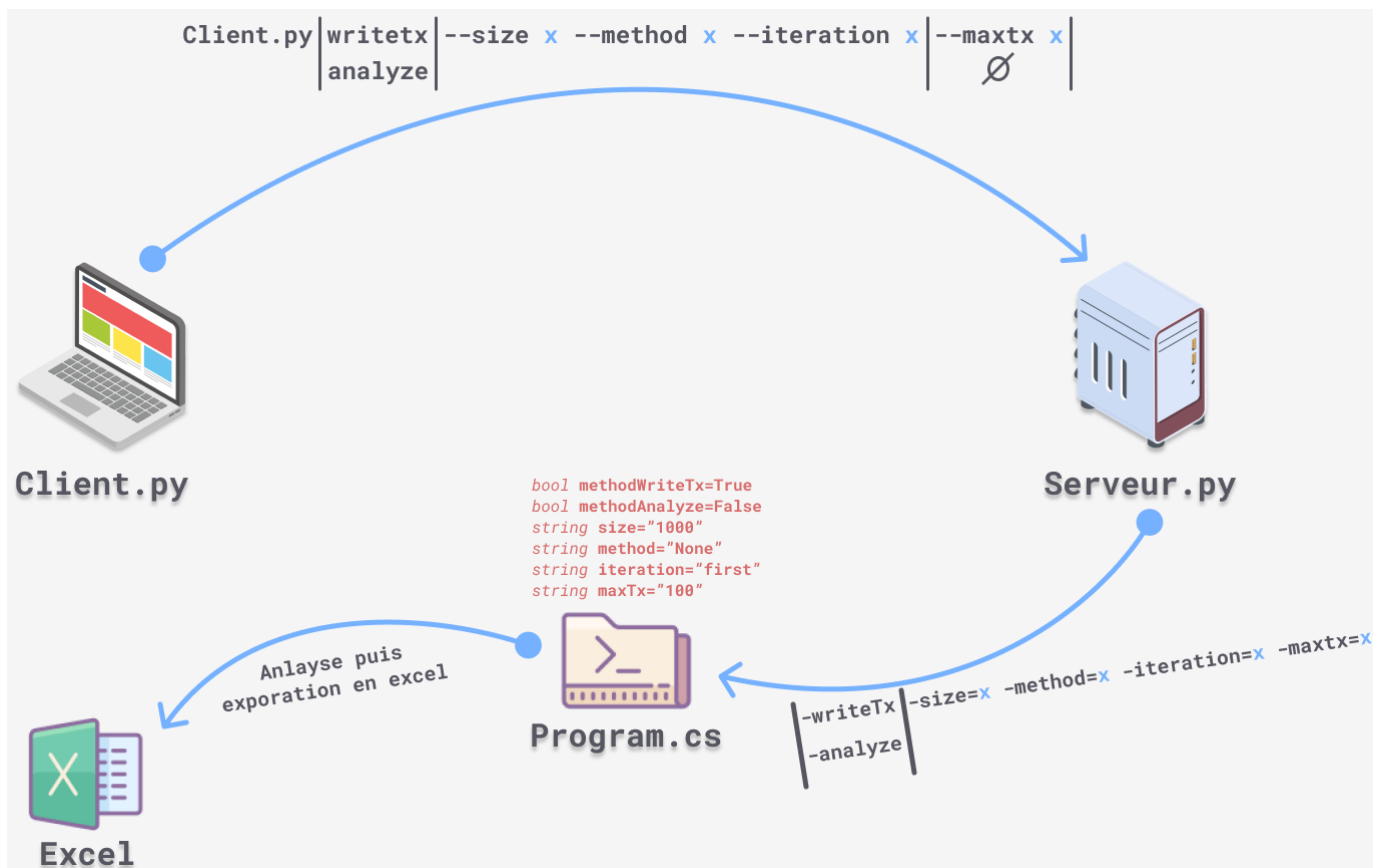
• Connection Client/Serveur

Maintenant que notre programme peut analyser et écrire nos transactions, il faudrait que tout ceci puisse être automatisé car pour le moment nous avons deux fonctions distinctes dans notre programme principal, mais comment savoir laquelle appeler ?

Pour cela nous allons utiliser des arguments que nous allons passer au lancement de notre exécutable. C'est une notion qui m'était jusqu'à présent inconnu, mais lors de la création du fonction « main() », elle prend un tableau d'arguments en paramètre. Il sera donc de notre devoir lors de l'envoi de la commande de spécifier quel paramètre on veut ajouter à notre commande. Cela permet un peu comme un input, de faire interagir l'utilisateur avec le code, mais cette fois-ci, on peut tout lui donner directement, ce qui permet de rendre la chose facilement automatisable. Par manque de temps et étant donné que cela ne faisait pas partie intégrante des compétences qu'il m'était demandé d'acquérir pendant le stage, on a pu à l'aide de mon maître de stage développer rapidement une connexion client-serveur afin que l'ordinateur de contrôle et l'ordinateur cible puissent communiquer. De ce principe, on peut facilement envoyer depuis notre client la commande qui sera reçue par le serveur sur l'ordinateur cible, serveur qui lancera l'exécutable avec les paramètres demandés. J'ai pu apprendre à utiliser un outil pour réaliser des clients proprement avec différentes méthodes et commandes. Ce dernier se prénomme « **click** » il est lui aussi disponible sur le GitHub et la syntaxe afin de rajouter une option se note comme ceci :

```
Client.py | writetx | --size x --method x --iteration x | --maxtx x |
          | analyze |                                     | Ø |
```

Si on doit résumer ce à quoi peut ressembler notre système contenant la connexion client-serveur ainsi que l'interaction avec notre programme d'écriture, cela pourrait ressembler à ceci :



Voici donc ce qu'il va se passer quand notre ordinateur client va lancer cette commande. Le serveur va tout simplement la découper pour en récupérer les informations importantes et les passer en paramètre à notre programme pour qu'il sache s'il doit lancer une analyse ou une écriture.

c) Automatisation.

Maintenant, qui dit test dit aussi automatisation, puisque plus le nombre de données est important à traiter plus il sera fatigant d'attendre pour que tout se fasse, il est important de trouver un moyen d'y remédier. Sans oublier le fait qu'il faille aussi effectuer plusieurs actions telles que lancer l'écriture, redémarrer l'ordinateur en envoyant un signal à l'Arduino, puis taper la commande analyse après que l'ordinateur s'est redémarré. Bref, tout ceci si on le lançait à la main pourrait nuire à nos tests du fait que l'ordinateur ne serait jamais redémarré au même moment.

Pour y remédier, mon maître de stage m'a proposé une solution : **Robot Framework** et son robot **Gherkin**. Robot Framework est une extension de Vscodé permettant grâce à de nombreuses bibliothèques d'automatiser un peu toutes les tâches telles qu'écrire sur un port série, ou bien taper une commande dans le CMD. Le Gherkin quant à lui est une sorte de manière d'écrire des scénarios en langage naturel grâce à des mots-clés.

Dans un premier temps, il suffit de créer un fichier « .robot », ensuite on règle nos « settings », c'est-à-dire que l'on met en place nos librairies :

```
*** Settings ***
```

```
Library    SerialLibrary
```

```
Library    OperatingSystem
```

SerialLibrary comme sont l'indique permet d'effectuer des opérations sur un port série, comme s'y connecter, envoyer des données, ...

Quant à **OperatingSystem**, cette librairie permet d'exécuter des commandes ce qui nous permettra de lancer automatiquement les commandes du client. Maintenant pour commencer à écrire nos scénarios il faut tout mettre dans une catégorie « test cases » :

```
Scenario: Power Failure Test --> size: 1ko & WriteThrough method
```

```
    Given I connect to COM4 with baudrate 9600
```

```
    When I start to write with a size of 1ko and with the WriteThrough method and with the iteration first and with 200 maxtx
```

```
        And I restart the computer
```

```
        And I wait for the computer to restart
```

```
    Then I can analyze the corrupted files with a size of 1ko and with the WriteThrough method and with the iteration first
```

Il suffit donc de donner un nom à notre Scenario afin que l'on puisse s'y retrouver. Comme on peut le voir ici, j'ai pu faire un scénario pour différentes tailles de transactions ainsi que différentes méthodes d'écriture. Comme on peut le voir sur le code ci-dessus, c'est de cette manière que le Gherkin opère :

- **Given** : Permet de donner un contexte initial à la création de notre scénario. Dans notre cas, il nous permet de dire que l'on va se connecter au port série de l'Arduino par le port 4 avec un débit de 9600 bytes/s.
- **When** : Sert à donner les événements à exécuter dans notre cas, l'écriture des transactions, sachant que **And** permet d'ajouter d'autres actions à effectuer à la suite de celle-ci.
- **Then** : Quant à lui, ce dernier mot clé nous donne l'action à effectuer à la toute fin, ici l'analyse de notre cycle puis sa conversion dans le fichier Excel.

Une chose que vous avez pu remarquer c'est qu'on ne puisse pas juste marquer « I restart the computer » pour qu'il redémarre l'ordinateur, ce serait de la magie. On appelle ça un « keyword », c'est l'un des fonctionnements principaux de Robot Keyword. On met le nom de notre mot-clé puis ce qu'il exécute juste en dessous, cela permet de garder un programme principal qui est facilement lisible. Voici à quoi ils ressemblent :

```
*** Keywords ***
```

```
I connect to ${port} with baudrate ${baudrate}
```

```
    Connect    ${port}    ${baudrate}
```

```
    Sleep      2
```

```
I start to write with a size of 1ko and with the WriteThrough method and with the iteration ${iteration} and with ${maxtx} maxtx
```

```
    Run        python C:\\Users\\guillotin\\_work\\git\\poweroffrobot\\src\\Test_Client\\client.py writetx --size 1000 --method WriteThrough --iteration ${iteration} --maxtx ${maxtx}
```

```
    Sleep      30s
```

```
I restart the computer
```

```
    Write      RESTART
```

```
I wait for the computer to restart
```

```
    Sleep      50s
```





On voit donc bel et bien que cela est tout de suite moins visible malgré les librairies qui nous facilitent la vie.

Analyse des Résultats/

Nous savons désormais comment tout se passe pour écrire nos transactions, de l'automatisation des scénarios à l'analyse de nos résultats. Voyons maintenant ce que nous pouvons faire avec ces derniers et essayons de trouver des solutions pour prévenir la corruption de fichiers lors de coupures de courant.

Nous avons dans un premier temps fait plusieurs scénarios. Pour chacun d'entre eux, nous faisons tourner 10 cycles afin d'avoir un certain nombre de données à traiter. Voici les différents tests qui ont été réalisés :

Possible modifications:

 size	 maxTx	 storage/OS	 Writing method
1ko	200 dossiers	HDD/Win7	WriteThrough
100ko	100 dossiers	SSD/Win10	None
10Mo	50 dossiers		Asynchronous
50Mo	25 dossiers		Kernel32
100Mo			WriteTrough

d) HDD/Win7

- 1/100ko, 100 dossiers – WriteThrough, ...

Commençons par parler des résultats obtenus sur le HDD muni de Windows 7 (les résultats sont disponibles dans le fichier Excel fourni dans le GitHub). On peut voir que dans un premier temps, pour les petits fichiers allant jusqu'à 100ko pour une écriture de type **WriteThrough** ou autres, on a essayé de réaliser des cycles de 100 transactions maximum. Le problème c'est vite montré : l'écriture des fichiers se faisant trop rapidement (environ 140ms/tx), à peine les fichiers écrits, ils sont directement remplacés. Ce qui au moment de la coupure peut provoquer un très grand nombre de fichiers corrompus. De ce fait on a pu

observer sur les dix cycles une moyenne de 27 transactions corrompus, ce qui sur 100 fait à peu près 1/3 des dossiers. Cette valeur n'est donc pas à prendre à la légère et nécessite alors un autre test qui sera d'augmenter le nombre maximal de transaction dans un cycle à 200 pour si cela change, car dans l'état actuel des choses, Glory ne peut se permettre perdre autant de données lors d'une coupure de courant.

- **1/100ko, 200 dossiers – WriteThrough**

La première chose que l'on peut constater c'est que nous passons maintenant à environ 8 fichiers corrompus sur 200 à la place de 27 sur 100, ce qui est colossal. On a diminué d'un facteur trois le nombre d'erreur juste en augmentant le nombre de transactions. Il faut noter sur ce fait qu'il n'est visible uniquement pour de petites tailles. Car plus l'on monte haut, plus les fichiers mettent de temps à s'écrire et moins le fait qu'ils s'effacent impactera les autres. On peut aussi constater que même si je mets 50, 100 ou même 200 transactions max, on peut voir un nombre de fichier généralement qui est à 1 de plus (51, 101 ou 201). Ceci est dû au fait qu'avant d'effacer le plus ancien, on va d'abord écrire le nouveau, ce qui peut ne pas se supprimer si la coupure de courant arrive au mauvais moment.

- **1/100ko, 200 dossiers – None**

Voyons voir maintenant si utiliser une autre méthode d'écriture peut changer les choses. Il est important de noter que ces valeurs sont à prendre à la légère du fait que par manque de temps, seulement 10 cycles ont pu être réalisé par scénario, ce qui reste assez restreint comme échantillon. Si on avait voulu des résultats plus viables proche de la perfection, il aurait fallu réaliser des centaines de cycle. Cependant, en se fiant à ce que nous avons pu trouver lors de l'écriture avec la méthode None qu'encore une fois on divise par deux le nombre d'erreurs pour se retrouver à seulement 4 transactions corrompus.

- **1/100ko, 200 dossiers – Asynchronous**

Rien d'intéressant à noter, les résultats étant très proches de ceux de None, on se place aux alentours de 4 transactions erronées. Une chose à noter serait au niveau des temps d'écriture, y a-t-il une différence ? malheureusement non, en tout cas rien de notable cela se joue à 5 millisecondes de différences ce qui n'est pas un espace assez conséquent pour que l'on puisse en ressortir un résultat pertinent.

- **10Mo, 100 dossiers – WriteThrough, ...**

Pour les plus gros fichiers de 1Mo, voyons ce que cela donne pour tous les types d'écriture. On peut voir qu'on arrive déjà à 2,139 secondes et 1,2 Dossiers corrompus pour le **WriteThrough**. Pour ce qui est de la méthode **Asynchronous**, on passe tout de suite sur 1,984 secondes ce qui accélère déjà un peu le processus, cependant, pas de changements quant à quantité de dossiers corrompus. Quant au dernier type : **None**, nous atteignons 1,8 dossiers corrompus et une vitesse moyenne d'environ 2,049 secondes. On peut en conclure rapidement que la méthode Asynchronous performe un peu mieux que ses concurrents sur les fichiers de moyennes tailles.

- **50Mo, 50 dossiers – WriteThrough, ...**

Passons maintenant à des volumes de fichiers assez imposant. Si dans la partie précédente, nous avons pu observer une augmentation du temps d'écriture de x10 environ, ce qui finalement reste assez peu comparé à l'écart de volume pour passer de 100ko à 1Mo. Tandis que pour passer de 10Mo à 50Mo, c'est x5 sur la durée, nous sommes ici à 9,261 secondes pour une transaction ce qui est finalement moins que pour passer de 100ko à 10Mo. Il peut donc être très intéressant de passer sur des fichiers plus gros car le rapport quantité/temps est plus intéressant. Voyons ce qu'il en est des autres méthodes :

Méthode d'écriture	Dossiers corrompus	Temps moyen d'une Tx(s)
WriteThrough	1	9,261
None	1	9,758
Asynchronous	1,2	9,432

- **100Mo, 25 dossiers – WriteThrough, ...**

Voici la plus grande taille de fichier que nous pouvons tester. Il est obligatoire de changer le nombre maximum de transaction du fait que pour les gros fichiers, si l'on avait choisi 200 transactions, le temps que le test se finisse aurait été trop long. Il a donc de cette manière fallu raccourcir un peu le tout. Voyons voir maintenant les résultats.

Méthode d'écriture	Dossiers corrompus	Temps moyen d'une Tx(s)
WriteThrough	1	18,489
None	1	18,804
Asynchronous	1,1	18,751

Encore une nouvelle fois, la méthode « **WriteThrough** » semble avoir devancé les autres en termes de temps d'écriture, même si nous sommes d'accord, ceci n'est pas très colossal (on ne parle même pas d'une seconde de plus ou de moins). Une chose à noter serait que l'on peut voir qu'il ne nous reste toujours qu'un dossier corrompu à la fin. Cela est dû comme dit précédemment au fait que notre programme écrit une transaction et attend qu'elle soit finie avant de supprimer la plus ancienne et comme le programme est toujours en train d'écrire, alors il est presque sûr que à chaque cycle au moins une transaction soit abîmée, sinon il faudrait que la coupure survienne au moment où la transaction ai fini de s'écrire sans en relancer une autre.

En bref sur ce type de stockage ci, on peut en conclure que plus on augmente la taille des fichiers, moins il y a de chance que nos transactions soient corrompues. On peut voir que ce qui ressort de plus rentable au niveau du temps et des dossiers corrompus serait d'écrire nos fichiers avec une taille de 10Mo minimum. De ce fait ci, nous sommes presque tout le temps sans aucune faute, si l'on oublie celle qui est en train de s'écrire et de plus l'écriture reste assez rapide : on ne parle qu'en une poignée de secondes (une moyenne de 2). Cependant et nous le verrons un peu plus tard, nous ne pouvons-nous permettre de conclure sur les types d'écriture puisqu'ils ne font pas drastiquement changer les valeurs et nos différences peuvent donc être dû à de simple variations de performance de l'ordinateur ce qui arrive souvent.

e) SSD/Win10

Changer de milieu en passant sur un SSD (solid state drive) avec le système d'exploitation win10 devrait nous permettre d'observer de vrai changement quant aux nombres de fichiers corrompus, mais surtout au niveau de la vitesse d'écriture, puisqu'il est connu que grâce à leur mémoire flash ils peuvent écrire jusqu'à 7450 Mb/s contre seulement 200 Mb/s pour un HDD. Cette différence colossale devrait donc changer pas mal de chose, mais cela saura-il en justifier le prix ?

• Fausses méthodes d'écritures

On a pu précédemment voir que l'on pouvait choisir entre différentes méthodes d'écriture afin de voir si cela aurait un quelconque impact sur nos résultats. Ces options d'écriture sont une fonctionnalité de **FileStream**, une bibliothèque de Microsoft permettant d'écrire des fichiers. Cependant en regardant plus précisément en faisant du « monitoring » (méthode permettant de voir ce que notre programme appelle comme fichier jusqu'au plus bas niveau afin de mieux comprendre ce qu'il se passe). On a donc pu observer que notre méthode `writeFile` appelé une fonction plus bas niveau du même nom dans une bibliothèque s'appelant **Kernel32**. En observant les paramètres qui étaient confiés à cette fonction, aucune ne mentionnait nos méthodes d'écriture. Excepté une : `WriteThrough`, cependant même si nous mettions `WriteThrough` en option, rien ne se passait.

A l'aide de mon maître de stage, nous avons donc pu créer une autre méthode dans notre programme qui appelle explicitement cette fonction avec le paramètre « `WriteThrough` » afin de voir si cela a cette fois ci changé quelque chose.

• 1/100ko, 200 dossiers – No WriteThrough/WriteThrough

Méthode d'écriture	Taille(kilo octet)	Dossiers corrompus	Temps moyen d'une Tx(ms)
No WriteThrough	1	0,3	163
WriteThrough	1	0,3	163
No WriteThrough	100	0,2	255
WriteThrough	100	0,2	265

Pas de changements symboliques entre les deux valeurs excepté le fait que comparé au HDD, on arrive cette fois ci à être beaucoup plus précis sur l'écriture de fichiers très petit grâce à ce nouveau type de mémoire qui change du disque rotatif du HDD. On se retrouve donc quelquefois à réussir d'écrire la dernière transaction en entière avant qu'il y ait la coupure de courant.

• 10Mo, 100 dossiers – No WriteThrough/WriteThrough

Méthode d'écriture	Dossiers corrompus	Temps moyen d'une Tx(ms)
No WriteThrough	0,9	565
WriteThrough	0,7	299

La différence commence à être flagrante puisque la différence de temps entre utiliser WriteThrough ou non fait évoluer par deux le résultat. On peut déjà en déduire que l'impact à ce niveau-là va en faveur de l'utilisation de cette méthode.

- **50Mo, 50 dossiers – No WriteThrough/WriteThrough**

Méthode d'écriture	Dossiers corrompus	Temps moyen d'une Tx(ms)
No WriteThrough	1	2515
WriteThrough	0,8	974

Il est possible de voir que à partir de plus gros fichier, comme vu précédemment, on se retrouve à stagner autour de 1 fichier corrompu qui est celui en train de s'écrire. Quant au temps moyen, on peut encore une fois voir que plus on monte en volume de transactions, plus la méthode WriteThrough à son importance.

- **100Mo, 25 dossiers – No WriteThrough/WriteThrough**

Méthode d'écriture	Dossiers corrompus	Temps moyen d'une Tx(ms)
No WriteThrough	0,7	4386
WriteThrough	0,8	2386

Une chose est encore prouvée ici, c'est bien que plus la taille des fichiers augmente, plus la méthode WriteThrough a son importance et nous permet de gagner un temps considérable sur l'écriture de nos fichiers, car environ 2 secondes par transactions ça ne paraît pas beaucoup, mais reporté sur des centaines, cela peut faire économiser de précieuses minutes.

Pour conclure sur cette partie SSD, on voit bien qu'il est désormais possible de gagner un temps énorme car la vitesse d'écriture de ce dernier le permet, sans oublier que sa mémoire flash lui permet contrairement au HDD d'être plus précis et stable dans son écriture ce qui permet d'avoir surtout sur les petits fichiers des cycles sans aucune perte, ce qui n'est pas négligeable.

- **Conclusion des résultats**

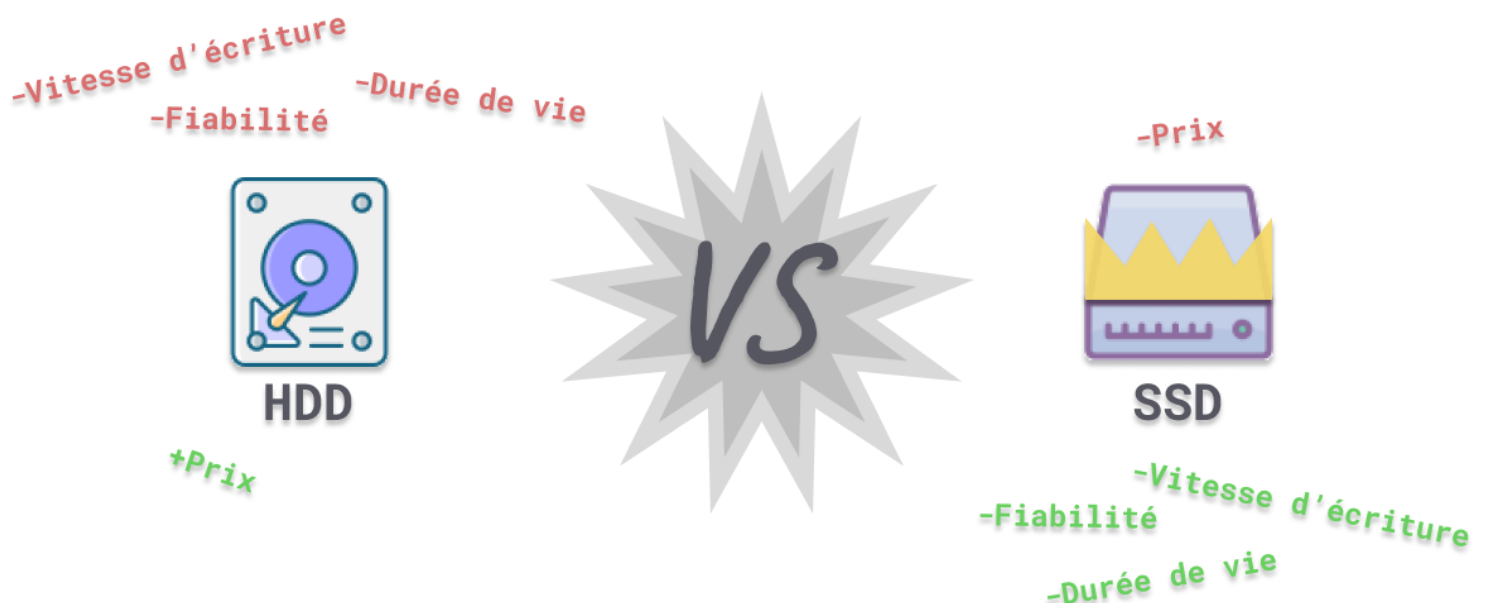
Si l'on doit bien ressortir une seule chose de ces tests, c'est bien qu'autant au niveau performance que nombre de dossiers corrompu, le SSD est loin devant. Car même si sur les fichiers de petite taille, la rapidité reste approximativement identique (de l'ordre de dizaines de millisecondes), nous pouvons être sûr d'une chose, le SSD est bien plus fiable du fait que les cycles varient entre 1 et 0 erreurs, tandis que le HDD nous sort une moyenne de 8 transactions corrompus avec des cycles parfois à plus de 10 dossiers erronés.

La différence du temps se fait à partir de fichiers de 10 Mo, à partir de là aucun doute ne fait que le SSD fasse gagner du temps, et surtout si l'on prend l'option WriteThrough. Le HDD est au moins 10x plus long sur les fichiers de 50Mo et 100Mo, tandis que sur ceux de 10Mo, on est aux environs d'une différence de facteur 5.

Cependant, quand est-il de la question de rentabilité. Sur le marché actuel, entre un disque dur HDD et un Solid State drive, il existe un facteur 2 de différence de prix entre les deux, c'est-à-dire que pour 1To de HDD il faudra déboursier environ 50€ tandis que pour un SSD, on se rapproche généralement des 100€. De ce fait là, il reste à voir si cette différence de prix qui n'est pas négligeable vaut la différence de performance qu'elle apporte. Comme on a pu le voir précédemment, en matière de rapidité, il n'y a pas de débat sur le fait que le SSD est plus rapide. Il est même jusqu'à 10 fois plus rapide. Sur ce point-là, le SSD remporte un point. Pour ce qui est maintenant de la fiabilité quant à la corruption des transactions, un problème se pose. Certes le SSD effectue un travail remarquable sur les fichiers de petite taille jusqu'à 100Ko, cependant après ça, on ne ressent presque aucune différence : le dernier fichier en train de s'écrire se corrompt et on s'arrête ainsi, aucun autre ne l'est. Ce n'est donc pas ce facteur qui changera la donne. Un autre facteur quant à lui peut faire pencher la balance : la durée de vie. Etant donné qu'un automate bancaire est appelé à durer dans le temps, il serait malvenu qu'au bout d'un certain temps, un stockage perde en performances ce qui pourrait nuire aux données et amener sur des frais supplémentaires.

De ce fait on sait qu'un HDD a une durée de vie de 3 à 5 ans dû à son disque rotatif qui est mené à s'épuiser dans le temps, tandis qu'un SSD peut durer jusqu'à 10 ans en moyenne par le fait que sa mémoire flash lui permette de mieux gérer l'espace et de moins s'épuiser puisque tout est électronique. Donc sans compter en plus le fait que les HDD sont plus fragiles, plus lourd et j'en passe, font que tout nous mène à dire que pour un prix de seulement 2x plus chère, le SSD est de loin plus rentable que son concurrent dans tous les domaines. Il est donc conseillé à la suite de cette analyse de migrer vers ce nouveau type de stockage car il convient pour toutes tailles de fichiers mais surtout pour les fichiers de petite taille du fait de sa fiabilité et de sa rapidité.

Voici un schéma permettant de mieux visualiser ce que nous avons pu dire jusqu'à présent, ce qui clôturera par ailleurs nos analyses.



Retours sur le stage/

a)Expérience humaine.

En tant que stagiaire pour la première fois dans le vrai monde du travail, on se rend vite compte qu'il n'y a aucun point commun avec l'école. En rentrant la première fois, on se fait accueillir à l'accueil, puis une personne vient vous faire visiter les locaux ainsi que les personnes y travaillant. On vous amène ensuite récupérer un ordinateur pour que le travail commence. Le premier point qui a pu me surprendre, c'est tous ces systèmes de sécurité informatique mis en place sur l'ordinateur. Il y a tout un tas de démarches avant d'accéder à notre session mais heureusement que j'ai pu me faire accompagner par les personnes chargées du pôle informatique de Glory. Si l'on continue par-là, j'ai aussi pu être surpris du fait de la répartition des équipes au sein de Glory. On y retrouve différents espaces au sein desquels on y retrouve les équipes : commerciales s'occupant des ventes, l'équipe du pôle informatique, l'équipe communication, mon équipe : l'équipe solutions, comprenant les développeurs chargés de différents projets sur les automates et des supports qui s'occupent tant du dépannage software que hardware et ce autant en France que à l'étranger.

J'ai pu comprendre que travailler à Glory c'est avant tout un esprit qui relie toutes ces équipes en une seule dans le but de continuer à faire avancer cette entreprise le plus loin et longtemps possible.

b)Difficultés rencontrées

Durant mon stage j'ai pu rencontrer différentes problématiques qu'il a fallu résoudre. Autant sur la partie électronique qu'informatique. Des pièges ont aussi été dissimulés par mon maître de stage, ce qui m'a permis d'essayer de les résoudre afin de monter en compétence. Sur la partie Arduino, nous étions confrontés à plusieurs problèmes. Quand on allumait le microcontrôleur, celui-ci faisait clignoter la LED. Ceci n'était enfaite seulement dû au fait que la pin 13 de l'Arduino est relié à la LED d'état de cette dernière. Ce qui signifie que quand ce dernier s'allume, la LED d'état clignote (variation de tension) ce qui signifie que tous ce qui est branché sur ce port clignotera aussi.

Second problème que j'ai pu énoncer précédemment qui était bien plus complexe. Le fait qu'une connexion au port Série fasse redémarrer la carte Arduino. Il fallait donc trouver un moyen de garder du courant passant pour que même si l'Arduino ne fournit plus de courant, on se retrouve tout de même avec l'ordinateur qui ne redémarre pas.

En passant maintenant sur le code, il y avait d'abord la problématique qui était de savoir comment nous allions structurer le code, comment fonctionne le programme actuel de l'équipe solution (atmlp) et comment analyser nos données. Tous ces problèmes m'ont permis de me surpasser pendant ce stage et de trouver un but, un objectif à atteindre, ce qui rendait le travail bien plus intéressant. Les horaires de travail étant aussi assez flexibles, il a fallu adapter son emploi du temps en autonomie pour faire preuve d'efficacité et gérer la quantité de travail qu'il fallait abattre.

Conclusion/

Afin de tout de même clôturer ce rapport qui met fin à mon stage, même si cela est triste de devoir quitter l'entreprise et l'équipe qui a su m'accueillir avec tant de bienveillance et de pédagogie, je tiens à tous les remercier pour cette expérience qui me sera très utile autant pour les 2 années scolaires à venir que pour ma future vie professionnelle. Je tiens aussi à remercier Polytech Dijon de permettre aux élèves de partir en stage afin de monter en compétence, mais aussi mes maîtres de stage : Josselin Colas et Matthieu Picovski pour leur aide tout au long de ce stage.

Merci encore d'avoir pris le temps de lire ce rapport,

GUILLOT NATTHAN



MERCI!

[Lien menant au GitHub](#)