Imepro

Application de monitoring et d’aide des élèves

École : CIFOM, École technique

Atelier : Salle 207

Langage : C#

Auteur : Stan Gouvernon

Supérieur : Alain Jeanmaire

Date : 24.01.2023-22.03.2023

Table des matières

[Présentation du projet 3](#_Toc134942461)

[Prérequis 3](#_Toc134942462)

[Matériel et logiciels 3](#_Toc134942463)

[Connaissances requises 4](#_Toc134942464)

[Cahier des charges 4](#_Toc134942465)

[Situation de départ 4](#_Toc134942466)

[But du projet 4](#_Toc134942467)

[Maquettes des applications 4](#_Toc134942468)

[Élève 4](#_Toc134942469)

[Professeur 4](#_Toc134942470)

[Fonctionnalités prévues 5](#_Toc134942471)

[Élève 5](#_Toc134942472)

[Professeur 5](#_Toc134942473)

[Communication 6](#_Toc134942474)

[Schéma de fonctionnement 6](#_Toc134942475)

[Schéma de classe 7](#_Toc134942476)

[Guide d’utilisateur 10](#_Toc134942477)

[Professeur 10](#_Toc134942478)

[Élève 10](#_Toc134942479)

[Gestion du projet 11](#_Toc134942480)

[Mise en place 11](#_Toc134942481)

[Matériel 11](#_Toc134942482)

[Logiciel 11](#_Toc134942483)

[Application Élève 11](#_Toc134942484)

[Interface 11](#_Toc134942485)

[Démarrage 12](#_Toc134942486)

[Fonctionnement 12](#_Toc134942487)

[Spécialisation 12](#_Toc134942488)

[Navigateur Intégré 12](#_Toc134942489)

[Réception de messages 12](#_Toc134942490)

[Réception du Stream 12](#_Toc134942491)

[Fermeture 13](#_Toc134942492)

[Application Professeur 13](#_Toc134942493)

[Démarrage 13](#_Toc134942494)

[Fonctionnement 13](#_Toc134942495)

[Fermeture 13](#_Toc134942496)

[Liaisons entre les deux applications 13](#_Toc134942497)

[Tests 13](#_Toc134942498)

[Méthode de test 13](#_Toc134942499)

[Tests Unitaires 14](#_Toc134942500)

[Protocole de Test 14](#_Toc134942501)

[Problèmes et solutions 14](#_Toc134942502)

[TreeView customisé 14](#_Toc134942503)

[IP au démarrage 15](#_Toc134942504)

[Affichage des miniatures 15](#_Toc134942505)

[Retirer les ListBox de la classe Élève 15](#_Toc134942506)

[Intégration du projet Navlogeur. 15](#_Toc134942507)

[La classe ne peux pas fonctionner. 15](#_Toc134942508)

[WebView2 n’est pas complet 15](#_Toc134942509)

[Empêcher l’utilisation du clavier avec des Hook. 15](#_Toc134942510)

[Références 16](#_Toc134942511)

[Webographie 16](#_Toc134942512)

[Glossaire 16](#_Toc134942513)

[Bilan 16](#_Toc134942514)

[Améliorations possibles 16](#_Toc134942515)

[Objectifs atteints 16](#_Toc134942516)

[Client 16](#_Toc134942517)

[Maître 16](#_Toc134942518)

[Communication 16](#_Toc134942519)

[Conclusion 16](#_Toc134942520)

# Présentation du projet

## Prérequis

### Matériel et logiciels

3 ordinateurs avec Windows 10.

Les trois ordinateurs sont connectés sur le même réseau local.

Visual Studio 2022 sur l’un des ordinateurs.

### Connaissances requises

Connaissances en développement orienté objet en C#

Connaissances sur l’envoi de données sur le réseau avec les Socket.

Connaissances sur le fonctionnement des applications.

Connaissances en multithreading.

## Cahier des charges

### Connexion

Il y a une application maître et une application client (élève)

Maître : affiche son adresse IP

Elève : permet la saisie de l’IP du maître et la stocke de manière pérenne. Reset du stockage possible

Elève : se souvient des IP précédentes et les propose en fonction de la demi-journée courante

Maître : affiche la liste des élèves connectés et met en évidence ceux possiblement déconnectés

Elève : répond aux sollicitations du maître (screenshot, liste ou kill de processus, Stream, contrôle...)

Elève/maître : annonce sa déconnexion en cas de fermeture, stoppant les processus de dialogue

Maître : peut mettre fin à une connexion (ex. si l’élève a laissé tourner et n’est pas dans la leçon)

### Interface graphique (IHM) principale du maître

L’IHM doit être organisée pour montrer le maximum d’information utile sans changer de fenêtre

Par ex, un TreeView avec les postes connectés à gauche et une case à cocher pour les observer

A droite, une mosaïque de captures des écrans, redimensionnée pour voir tous ceux cochés

Les miniatures sont accompagnées du nom du poste et de l’âge de la capture

Un slider de zoom permet d’agrandir les miniatures devenant visible par à une barre de défilement

On peut déployer ou replier les branches du TreeView, une par une ou toutes d’un coup

Le TreeView déployé montre les applications lancées et leur titre (après filtrage, voir filtre plus loin)

Pour les navigateurs, il est possible d’afficher l’historique du jour : heure, titre et si possible url

Il est possible de tuer le processus d’une application ou de l’ajouter au filtre des apps à ignorer

Il est possible d’ouvrir une fenêtre d’observation d’un poste en particulier. Elle est refermable.

Le dialogue entre le maître et les clients peut être consulté (historique et temps réel)

Le maître et les élèves peuvent s’envoyer des courts messages texte qui surgissent chez l’élève

Un bouton démarre la diffusion multicast sur les postes sélectionnés, un bouton la stoppe

Le multicast fonctionne sur n’importe quel PC. Pour le cast de 1 écran, on peut choisir lequel caster

On peut faire un multicast des deux écrans de l’enseignant, il s’affiche sur les deux écrans clients

Seul l’écran où se trouve la souris est réellement casté. L’autre affiche celui au départ de la souris

Si un client n’envoie plus d’informations, Une alerte est très visible dans le TreeView (rouge, durée)

### Filtrage de l’affichage des processus

Par défaut, le programme examine les nouveaux processus et ignore ceux présents à son lancement

Le maître diffuse trois listes d’applications : surveillées (voir), ignorées (cacher) et interdites (alerte)

Afin de compléter la liste, il est possible de désactiver le filtre.

Le maître diffuse une liste des expressions régulières dans le titre qui déclenchent une alerte

La liste des expressions régulières de titre est un simple fichier texte éditable sur le poste du maître

Le maître doit pouvoir envoyer une liste de sites autorisés (avec jokers) sur le même principe

Le client fait appliquer le filtre des sites (quitte à n’autoriser qu’un seul navigateur compatible)

### Capture d’écran

A intervalle régulier, la capture de tous les écrans des clients est envoyée et possiblement affichée

Le maître peut sauver l’image. Filename et texte dans l’image = yymmdd-hhmm-identité-Poste.png

### Diffusion multicast depuis le poste maître

Le maître peut diffuser de manière fluide sur les clients sélectionnés, avec le curseur souris.

C’est un streaming multicast (il n’y a qu’un flux pour tous les clients)

Le maître peut choisir la priorité du Stream : fenêtre, FullScreen, TopMost, TopMost et blocage du client

Il peut choisir le focus autorisé pendant le Stream : tout, rien, OneNote, DevEnv, VSCode, Word, etc...

### Streaming monoposte depuis le client, contrôle à distance

Depuis la fenêtre d’observation, il serait commode d’afficher un flux fluide de capture du client

Idéalement, le maître aurait le contrôle à distance sur le client pour apporter de l’aide (souris/clavier)

Pendant la prise de contrôle, l’enseignant choisit si élève peut (ou pas) utiliser sa souris et son clavier

L’affichage serait réactif (quand l’enseignant bouge la souris : pas de saccade, le curseur est visible)

Pendant le contrôle à distance, l’enseignant peut basculer entre l’écran 1, 2 ou les deux écrans

### Portabilité du client

Le client doit être livré sous forme de 3 projets. (1) une bibliothèque de classes contenant la logique (2) un démonstrateur utilisable : le client (3) un projet de tests unitaire (simule l’envoi de cmd maitre)

Le client doit détecter la configuration des adaptateurs Ethernet et proposer un script à copier-coller en PowerShell administrateur capable d’activer la bonne carte (celle sur le sous-réseau multicast)

La documentation GitLab doit indiquer comment intégrer la bibliothèque de classes dans un projet.L’intégration dans le projet Navlogeur sera faite par sa développeuse uniquement grâce à la doc.

Les mises à jour seront poussées sur GitLab et la développeuse suivra la doc pour les appliquer

Le client est prévu pour relayer des messages maitre↔client gérée par une méthode dédiée

Le navigateur de filtrage pourrait être Navlogueur (fourni avec documentation par la développeuse)

### Envoi de fichiers

Idéalement, il devrait être possible au maître d’envoyer des fichiers aux élèves sélectionnés

Idéalement, il devrait être possible aux clients d’envoyer des fichiers au maître

## Situation de départ

Pendant sa spécialisation, l’étudiant a appris à utiliser des sockets pour connecter des clients à un serveur et dialoguer avec lui. Il a réussi à envoyer la liste des processus, les titres des fenêtres Web et une capture des 2 écrans de l’utilisateur. Ces informations, gérées dans un thread séparé s’affichaient dans l’interface graphique de l’application du maître qui pouvait alors les consulter et, par exemple tuer un processus à distance. Un démonstrateur a été réalisé pour effectuer un multicast de l’écran de l’enseignant sur les postes clients. L’image est découpée en paquets non compressés, envoyés en UPD en multicast sur le sous-réseau. Les clients reconstituent l’image quand les paquets ont été reçus. Le multicast fonctionne moyennant une configuration correcte des cartes réseau. Le projet consiste à poursuivre ce développement de façon à disposer d’une alternative à Impero pour la surveillance des postes des élèves, la diffusion de démonstrations, la prise de contrôle pour effectuer une assistance à distance, le tout avec un logiciel léger et offrant des options non disponibles dans Impero, par exemple le Stream double écran et le filtrage pendant le Stream

## But du projet

## Maquettes des applications

### Élève

Principal

IP du professeur

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

### Professeur

Principale

Affichage individuel

Choix pour le Stream

Choix de l’IP

Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement

## Fonctionnalités prévues

### Élève

Prendre une capture d’écran de tous les écrans

Récupération des processus lancé par l’utilisateur.

Récupération de toutes les url ouvertes dans tous les navigateurs

Minimiser en TrayIcon pour cacher l’application.

Afficher l’écran de l’application maître

Afficher le Stream reçu.

Bloquer le clavier et la souris pendant un Stream.

Autoriser que certaines applications pendant un Stream.

Limiter les sites visitables à la liste reçue.

### Professeur

Arrête des processus indésirables des clients

Minimiser en TrayIcon pour cacher l’application.

Afficher les urls reçus dans un historique pour chaque navigateur.

Afficher les processus reçus.

Permettre la sélection des images que l’on veut afficher.

Afficher les captures d’écrans reçues de manière dynamique.

Afficher toutes les données dans un TreeView.

Permettre l’affichage d’un élève individuel.

Permet d’activer ou non un filtre couleur pour les processus et les urls.

Permet la capture d’un ou plusieurs écrans et leur Streaming en multicast.

Envoyer la liste des sites visitables.

Affichage modulable pour permettre la visibilité des informations.

### Communication

Envoi des données dans des threads séparé.

Transférer les données à haut débit.

Envoi des données élèves au professeur.

Envoi des captures d’écran au professeur.

Envoi de la liste des sites autorisés à tous les élèves.

Partage d’écran du professeur en multicast.

## Schéma de fonctionnement

sequenceDiagram

Participant S as Selenium

Actor E as Eleve

Participant AE as ApplicationEleve

Participant AP as ApplicationProfesseur

Actor P as Professeur

E->>AE:Démmarage

AE->>E:IP de ton professeur ?

AP->>P:Affiche l'IP

P->>E:Donne l'IP à ses élèves

E->>AE:157.26.227.198

AE->>AP:Demande de connxion

AE->>S:Démmarage

loop Toutes les 2 secondes

AE->>S:Vérification de l'url

end

loop Toutes les 15 secondes

AP->>AE:Demande d'informations

AE->>AE:Récupération des données

AE->>AP:Envoi des informatinos

AP->>AE:Demande de la capture d'écran

AE->>AE:Prise de la capture d'écran

AE->>AP:Envoi de la catpute d'écran

end

loop When the teacher wants

P->>AP:Visualisation des informations

end

P->>AP:Démmarage du Streaming

AP->>AE:Préviens les élèves concernés

AE->>AE:Lance la réception

loop Jusqu'a ce que le professeur arret le partage

AP->>AE:Image

end

E->>AE:Arret

AE->>AP:Signalement de l'arret

AE->>S:Arret de selenium

P->>AP:Arret

AP->>AE:Signalement de l'arret

## Schéma de classe

classDiagram

class IpForTheWeek{

+String[] lundi

+String[] mardi

+String[] mercredi

+String[] jeudi

+String[] vendredi

+String[] samedi

+String[] dimanche

+SetIp(string) void

+GetIp() string

}

class Url{

+DateTime CaptureTime

+String Browser

+String Name

+ToString() String

}

class HistoriqueUrl{

+List~Url~ chrome

+List~Url~ firefox

+List~Url~ seleniumchrome

+List~Url~ seleniumfirefox

+List~Url~ opera

+List~Url~ edge

+List~Url~ safari

+List~Url~ iexplorer

+List~Url~ custom

+AddUrl(Url) void

+VerifiyUrl(List~Url~, Url) void

}

class GlobalKeyboardHook{

+List~Key~ HookedKeys

-IntPtr hHook

-KeyboardHookProc hookProc

-int WH\_KEYBOARD\_LL

-int WM\_KEYDOWN

-int WM\_KEYUP

-int WM\_SYSKEYDOWN

-int WM\_SYSKEYUP

+Hook() void

+Unhook() void

-HookCallBack(int,int,KeyboardHookStruct) int

}

class Data{

+String UserName

+String ComputerName

+HistoriqueUrl Urls

+Dictionary<int,string> Processes

+BitMap ScreenShot

}

class StreamOptions{

+Priority priority

+Focus focus

}

class Priority{

<<enumeration>>

Widowed

Fullscreen

Topmost

Blocking

}

class Focus{

<<enumeration>>

Everything

OneNote

VisualStudio

VSCode

Word

}

class DataForTEacher{

+Socket SocketToStudent

+int ID

+int NumberOfFailure

+List~Message~ Messages

+ToString() string

}

Data <|-- DataForTEacher

class DataForStudent{

}

# Guide d’utilisateur

## Professeur

## Élève

En tant qu’élève l’application n’est pas très complexe, elle démarre automatiquement mais reste cachée. Si vous voulez l’ouvrir vous pouvez cliquer sur le TrayIcon en bas à droit de votre écran. Cela vous permettra de lancer des navigateurs web tel que chrome ou Firefox.

Lorsque l’application s’ouvre toute seul vous pourrez y voir soit un Stream de votre professeur soit un message de votre professeur.

Dans le cas d’un Stream l’application peut forcer son affichage au premier plan ou bloquer votre clavier et votre souris. Ne vous inquiétez pas c’est normal c’est le professeur qui le choisi au moment de lancer le Stream.

La seule autre chose que vous pouvez voir dans l’application sont les interactions de votre application avec l’application du professeur.

Au démarrage l’application peut vous demander l’adresse IP de votre professeur si c’est le cas demandez-lui son IP, elle est affichée sur son application. L’adresse que vous entrez sera sauvegardé pour la réessayer la semaine prochaine durant la même demi-journée

Si l’application ne se connecte pas à l’application du professeur c’est surement que l’IP du professeur à changer et que l’IP sauvegardé n’est plus la bonne. Dans ce cas-là vous pouvez pressez sur le bouton « Changer d’IP »

Si l’application ne semble pas recevoir le Stream de votre professeur vos pouvez cliquer sur le bouton « Help » et suivre les instructions données.

# Gestion du projet

Pour la gestion de projet j’ai utilisé un développement agile, il s’agit d’une méthode de travail faite pour privilégier des rendus réguliers au client et des retours rapides sur ce que l’on a fait. J’ai donc fait plusieurs rendus pour les nouvelles fonctionnalités que j’avais développé. Chaque rendu est composé de plusieurs tâches qui ont un thème commun. Cela m’a permis de discuter des problèmes et des améliorations avec mon supérieur très rapidement pour ne pas perdre de temps.

# Mise en place

## Matériel

L’installation matériel est relativement simple il suffit que deux ordinateurs Windows soient sur le même réseau local. Je ne décrirais pas l’installation de Windows ici. Mais pour la liaison au même réseau local il suffit de connecter les deux ordinateurs au même switch.

## Logiciel

Installation de Visual Studio 2022. Mais elle nécessite tout de même des droits administrateurs. Mais si c’est le cas elle est simple, il vous suffit d’aller sur leur site <https://visualstudio.microsoft.com/fr/> puis de télécharger la version communautaire pour Windows. Á l’installation le seul module qui soit nécessaire est « Développement .NET Desktop ».

Il vous suffit ensuite de cloner le projet <https://git.s2.rpn.ch/stany24/tpi-impero-v2> ce qui nécessite de vous connecter avec vous identifiants de l’école.

# Application Élève

## Interface

PbxScreenShot : Permet d’afficher le Stream reçu par le professeur. La PictureBox est cachée tant qu’il n’y a pas de Stream. Ce qui permet de ne pas surcharger l’interface.

btnFirefox : Permet de lancer une instance de Firefox contrôlée par Sélénium.

btnChrome : Permet de lancer une instance de Chrome contrôlée par Sélénium.

lbxMessage : Cette ListBox permet d’afficher les messages reçus de professeur.

lbxConnection : Cette ListBox permet à l’élève de voir les communications entre lui et le professeur en directe.

## Démarrage

L’application essaye de charger l’IP du professeur depuis le fichier de configuration. Dans le cas ou l’application n’arrive pas à charger l’IP une interface permet à l’élève de rentrer l’IP de son professeur.

Puis un client est créé avec l’IP obtenue en dessus.

Puis deux tâches sont lancées en parallèles :

La connexion du client au maitre.

La vérification des navigateur lancés avec sélénium.

Après la connexion au professeur une nouvelle tâche est lancée, elle attend les demandes du professeur.

## Fonctionnement

### Spécialisation

### Navigateur Intégré

L’application élève permet de lancer des navigateurs intégrés avec Sélénium. Ils ont des avantages car nous pouvons les contrôler avec l’application cliente. Elle peut donc récupérer l’url actuel ainsi que d’interdire les url non autorisées en revenant en arrière.

### Réception de messages

L’application cliente est capable de recevoir des messages venant du professeur. Ils sont affichés dans une listbox pour avoir un semblant d’historique des messages.

### Réception du Stream

L’application cliente reçoit juste avant le flux d’image du Stream sa priorité et son focus autorisé.

Pour la priorité il y a 4 possibilités :

Windowed : l’application élève reste fenêtrée et n’a pas de caractéristique spéciale elle peut être minimisée ou cachée.

FullScreen : l’application élève est forcément en pleine écran mais elle ne peut pas être minimisée. D’autres applications peuvent tout de même se mettre devant.

TopMost : l’application élève est forcément en pleine écran ainsi que devant les autres applications elle ne peut pas non plus être minimisée.

Blocking : l’application est dans le même état qu’en TopMost mais l’élève n’a plus aucun contrôle sur le pc, ni avec la souris ni avec le clavier.

## Fermeture

L’application élève préviens l’application professeur qu’il s’arrête, puis il arrête tous les navigateurs lancés avec sélénium.

# Application Professeur

## Démarrage

Pour commencer l’application crée une instance de la classe MiniatureDisplayer d’après la taille du panel où elle va afficher les miniatures.

Puis l’application va chercher l’IP actuel de la machine, dans le cas ou la machine est sur plusieurs réseau une interface permet au professeur de choisir l’IP qu’il désire.

La prochaines tâche charge toutes les listes de configuration :

Processus ignorés au moment de l’affichage dans l’interface.

Processus alertés dans l’interface, signalé en rouge.

Urls alertés dans l’interface, signalé en rouge.

Urls autorisées dans les navigateurs Sélénium.

## Fonctionnement

### Spécialisation

### TreeView Détails

Le TreeView en haut à gauche permet de voir les informations des élèves. Les branches commencent par le nom de l’utilisateur puis le nom du pc. Ici la première branche se sépare en deux : les navigateurs et les processus. Dans la branche processus nous pouvons voir tous les processus lancés par cet utilisateur. Dans la branche navigateur nous pouvons voir les navigateurs lancés ainsi que les urls visités avec l’heure de visite.

### TreeView Miniatures

Le TreeView en bas à gauche permet de sélectionner quelles miniatures seront affichée dans le panel en haut à droite de l’écran.

Double cliquer sur un ordinateur permet d’ouvrir un affichage individuel.

### Affichage des Miniatures

Toutes les miniatures cochées dans le TreeView en bas à gauche seront affichée dans le panel. Leur taille peut être changée grâce au slider. Les miniatures sont mises à jour dès qu’une nouvelle image est reçue de l’élève. En dessous des miniatures on peux voir le nom du poste ainsi que le temps depuis la dernière mise à jour de l’image.

### Affichage individuel

Un seul affichage individuel ne peut être ouvert par élève. Il permet d’avoir un fenêtre entière dédiée à un élève ce qui permet de mieux voir ce que fait cet élève ou de lui envoyer des messages.

### Options de Streaming

Au démarrage d’un Stream le professeur peut choisir quelles élèves participeront ainsi que la Priorité et le Focus.

La Priorité du Stream définit :

Si l’application doit être en pleine écran.

Si l’application doit être devant toutes les autres.

Si l’utilisateur à le contrôle.

Le Focus de Stream définit dans le cas ou l’utilisateur à le contrôle quelles applications peuvent être ouvertes et utilisée pendant le Stream. Par exemple : tout, OneNote, Visual Studio, VSCode, Word, etc…

### Filtrage

Le filtrage des le TreeView Détails peut être activé ou non grâce au bouton « Activer/Désactiver » en haut à gauche de l’interface.

### Configuration

Processus Ignorés : Ces processus ne seront pas affichés dans le TreeView Détails

Processus Alertés : Ces processus seront affichés en rouge dans le TreeView Détails

Urls Alertés : Ces urls seront affichés en rouge dans le TreeView Détails.

Urls Autorisés : Ces urls seront bloqué des les navigateurs contrôlés par sélénium sur les ordinateurs des élèves.

Toutes les configurations sont stockées dans le des fichiers sur le disque.

## Fermeture

Á la fermeture l’application professeur préviens tous les clients qu’il s’arrête. Puis il sauve les listes de configuration dans des fichiers :

Processus ignorés au moment de l’affichage dans l’interface.

Processus alertés dans l’interface, signalé en rouge.

Urls alertés dans l’interface, signalé en rouge.

Urls autorisées dans les navigateurs Sélénium.

# Tests

## Méthode de test

Les tests ont été effectué de deux manières différentes : par des tests unitaires et par des tests visuels. Les tests unitaires sont très efficaces et permettent de faire beaucoup de tests en peu de temps, mais il faut les créer et certaines vérifications sont très difficiles à faire. C’est pourquoi dans ces cas-là j’ai préféré faire des tests visuels. Qui sont plus redondant mais aussi plus fiables dans certains cas.

## Tests Unitaires

## Protocole de Test

Brancher 4 ordinateurs sur le même réseau.

Démarrer les 4 ordinateurs.

Démarrer l’application professeur sur l’un des ordinateurs.

Démarrer l’application élève sur les 3 autres ordinateurs.

Regarder l’adresse IP du professeur sur l’application professeur.

Rentrer l’adresse sur les 3 applications élèves.

Vérifier la connexion au professeur sur les applications élèves « Connected ».

Lancer des navigateurs web sur les ordinateurs élèves.

Vérifier que les informations sont correctement affichées pour tous les élèves.

Vérifiez le bon fonctionnement de l’affiche des captures :

Redimensionnez la zone.

Ajouter/retirer des captures d’écran

Modifier le zoom.

Démarrer un Streaming en multicast en bloquant le clavier et la souris.

Vérifier que les élèves ne peuvent pas regagner le contrôle :

Essayer de déplacer la souris, de cliquer.

Essayer d’utiliser le clavier.

Essayer WINDOWS+L

Essayer CTRL+ALT+DEL

Arrêter le Stream :

Vérifier que les clients ont bien regagné le contrôle du clavier et la souris.

Lancer le Stream en TopMost et vérifier que les élèves ne peuvent pas mettre des fenêtres devant le Stream.

# Problèmes et solutions

## TreeView customisé

Afficher des casses à cocher dans un seul niveau du TreeView ainsi que des boutons dans le TreeView. Cela ne semble pas possible sans écrire ma propre fonction de génération d’un TreeView. Ce qui me prendrait beaucoup trop de temps. J’ai donc opté pour un deuxième TreeView qui permet juste la sélection.

## IP au démarrage

Au démarrage l’application professeur prenait la première adresse ipv4 quelle trouvait, j’ai donc ajouté une interface au démarrage pour choisir l’IP que l’on veut utiliser.

## Affichage des miniatures

La miniature et le label n’étaient visibles que dans une petite zone, car la miniature ainsi que le label sont une classe qui hérite de UserControl. De ce fait la classe à elle aussi une taille ce qui est à l’extérieur de cette taille n’est pas affiché.

## Retirer les ListBox de la classe Élève

On voudrait éviter d’avoir des éléments d’affichage dans la classe mais tous mes essais (BindingList, DataSource, Mise à jour manuelle) ont été futiles, car changer la source de donnée d’une ListBox (ou autres éléments d’affichage) depuis un thread qui ne s’occupe pas de l’interface est visiblement presque impossible car la ListBox n’accepte que les changements qui viennent du thread qui s’occupe de l’interface.

## Intégration du projet Navlogeur.

Le premier essai a été de transformer mon client Windows Forms en projet WPF. Cela s’est avéré très difficile car les versions utilisées de .NET ne sont pas identiques (Windows Forms <=4.7.2, WPF >=5.0). Le projet WPF ne pouvait pas utiliser les classes que j’avais créé lorsque je les lui donnais en référence. La solution à été d’ajouter la classe dans le projet WPF en tant que ressource liée.

## La classe ne peux pas fonctionner.

La classe ne peux pas fonctionner avec les deux using (System.Windows.Forms et System.Windows.Controls) qui sont tout les deux nécessaires pour le fonctionnement de Windows Forms et WPF. Cela m’a fait choisir d’arrêter d’essayer d’implémenter mon client en WPF

## WebView2 n’est pas complet

Pour avoir un navigateur intégré à mon client j’ai décidé d’utiliser une fenêtre WebView2. C’était fonctionnel mais il manquant des commodités de navigations : onglets, barre de recherche, bouton retour, etc... Finalement j’ai décidé d’intégrer Sélénium au projet car il offrirait un navigateur complet à tous les élèves et permet aussi de récupérer les urls.

## Empêcher l’utilisation du clavier avec des Hook.

Après avoir fais des recherches je suis arrivé sur ce projet : <https://www.codeproject.com/Articles/19004/A-Simple-C-Global-Low-Level-Keyboard-Hook> qui avait déjà implémenté le verrouillage du clavier. Mais après l’avoir testé dans mon projet une erreur revenait : Le delegate était perdu dans le Garbage Collector. Il a fallu que je modifie le code pour créer une instance du delegate dans une variable et ne plus l’instancier à chaque utilisation.

# Références

## Webographie

Cases à cocher uniquement à certains niveaux du TreeView : <http://dotnetfollower.com/wordpress/2011/05/winforms-treeview-hide-checkbox-of-treenode/>

Bouton dans le TreeView : <http://dotnetfollower.com/wordpress/2011/05/winforms-treeview-hide-checkbox-of-treenode/>

## Glossaire

TreeView : Affichage en forme de branche qui se sépare en plusieurs.

Stream : Vidéo en directe

Garbage Collector : Moyen de gérer la mémoire dans les langages de haut niveau.

# Bilan

## Améliorations possibles

## Objectifs atteints

### Client

### Maître

## Conclusion