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 5](#_Toc135060527)

[Prérequis 5](#_Toc135060528)

[Matériel et logiciels 5](#_Toc135060529)

[Connaissances requises 5](#_Toc135060530)

[Cahier des charges 5](#_Toc135060531)

[Connexion 5](#_Toc135060532)

[Interface graphique (IHM) principale du professeur 5](#_Toc135060533)

[Filtrage de l’affichage des processus 6](#_Toc135060534)

[Capture d’écran 7](#_Toc135060535)

[Diffusion multicast depuis le poste professeur 7](#_Toc135060536)

[Streaming monoposte depuis l’élève, contrôle à distance 7](#_Toc135060537)

[Portabilité de l’élève 7](#_Toc135060538)

[Envoi de fichiers 8](#_Toc135060539)

[Situation de départ 8](#_Toc135060540)

[Situation voulue 8](#_Toc135060541)

[But du projet 8](#_Toc135060542)

[Maquettes des applications 8](#_Toc135060543)

[Élève 8](#_Toc135060544)

[Professeur 9](#_Toc135060545)

[Fonctionnalités prévues 9](#_Toc135060546)

[Élève 9](#_Toc135060547)

[Professeur 10](#_Toc135060548)

[Communication 10](#_Toc135060549)

[Schéma de fonctionnement 10](#_Toc135060550)

[Schéma de classe 12](#_Toc135060551)

[Guide d’utilisateur 15](#_Toc135060552)

[Professeur 15](#_Toc135060553)

[Élève 15](#_Toc135060554)

[Gestion du projet 16](#_Toc135060555)

[Mise en place 16](#_Toc135060556)

[Matériel 16](#_Toc135060557)

[Logiciel 16](#_Toc135060558)

[Application Élève 16](#_Toc135060559)

[Interface 16](#_Toc135060560)

[Démarrage 17](#_Toc135060561)

[Fonctionnement 17](#_Toc135060562)

[Spécialisation 17](#_Toc135060563)

[Navigateur Intégré 17](#_Toc135060564)

[Réception de messages 17](#_Toc135060565)

[Réception du Stream 17](#_Toc135060566)

[Fermeture 18](#_Toc135060567)

[Application Professeur 18](#_Toc135060568)

[Démarrage 18](#_Toc135060569)

[Fonctionnement 18](#_Toc135060570)

[Spécialisation 18](#_Toc135060571)

[TreeView Détails 18](#_Toc135060572)

[TreeView Miniatures 18](#_Toc135060573)

[Affichage des Miniatures 18](#_Toc135060574)

[Affichage individuel 19](#_Toc135060575)

[Options de Streaming 19](#_Toc135060576)

[Filtrage 19](#_Toc135060577)

[Configuration 19](#_Toc135060578)

[Fermeture 19](#_Toc135060579)

[Tests 20](#_Toc135060580)

[Méthode de test 20](#_Toc135060581)

[Tests Unitaires 20](#_Toc135060582)

[Protocole de Test 20](#_Toc135060583)

[Problèmes et solutions 21](#_Toc135060584)

[TreeView customisé 21](#_Toc135060585)

[IP au démarrage 21](#_Toc135060586)

[Affichage des miniatures 21](#_Toc135060587)

[Retirer les ListBox de la classe Élève 21](#_Toc135060588)

[Intégration du projet Navlogeur. 21](#_Toc135060589)

[La classe ne peux pas fonctionner. 21](#_Toc135060590)

[WebView2 n’est pas complet 21](#_Toc135060591)

[Empêcher l’utilisation du clavier avec des Hook. 22](#_Toc135060592)

[Empêcher l’utilisation de certaines applications pendant le Stream 22](#_Toc135060593)

[Références 22](#_Toc135060594)

[Webographie 22](#_Toc135060595)

[Glossaire 22](#_Toc135060596)

[Bilan 22](#_Toc135060597)

[Améliorations possibles 22](#_Toc135060598)

[Objectifs atteints 22](#_Toc135060599)

[Élève 22](#_Toc135060600)

[Professeur 22](#_Toc135060601)

[Conclusion 22](#_Toc135060602)

# 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 professeur et une application élève.

Professeur : affiche son adresse IP

Elève : permet la saisie de l’IP du professeur 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

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

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

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

Professeur : 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 professeur

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 professeur et les élève s peut être consulté (historique et temps réel)

Le professeur 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 élève s

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

Si un élève 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 professeur 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 professeur 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 professeur

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

L’élève 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 élève s est envoyée et possiblement affichée

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

### Diffusion multicast depuis le poste professeur

Le professeur peut diffuser de manière fluide sur les élèves s sélectionnés, avec le curseur souris.

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

Le professeur peut choisir la priorité du Stream : fenêtre, FullScreen, TopMost, TopMost et blocage de l’élève.

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

### Streaming monoposte depuis l’élève, contrôle à distance

Depuis la fenêtre d’observation, il serait commode d’afficher un flux fluide de capture de l’élève.

Idéalement, le professeur aurait le contrôle à distance sur l’élève 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 de l’élève

L’élève doit être livré sous forme de 3 projets. (1) une bibliothèque de classes contenant la logique (2) un démonstrateur utilisable : l’application élève (3) un projet de tests unitaire (simule l’envoi de cmd professeur)

L’élève 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

L’élève est prévu pour relayer des messages professeur↔ élève 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 professeur d’envoyer des fichiers aux élèves sélectionnés

Idéalement, il devrait être possible aux élève s d’envoyer des fichiers au professeur

## Situation de départ

Pendant sa spécialisation, l’étudiant a appris à utiliser des sockets pour connecter des élèves 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 professeur 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 élèves. L’image est découpée en paquets non compressés, envoyés en UPD en multicast sur le sous-réseau. Les élèves reconstituent l’image quand les paquets ont été reçus. Le multicast fonctionne moyennant une configuration correcte des cartes réseau.

## Situation voulue

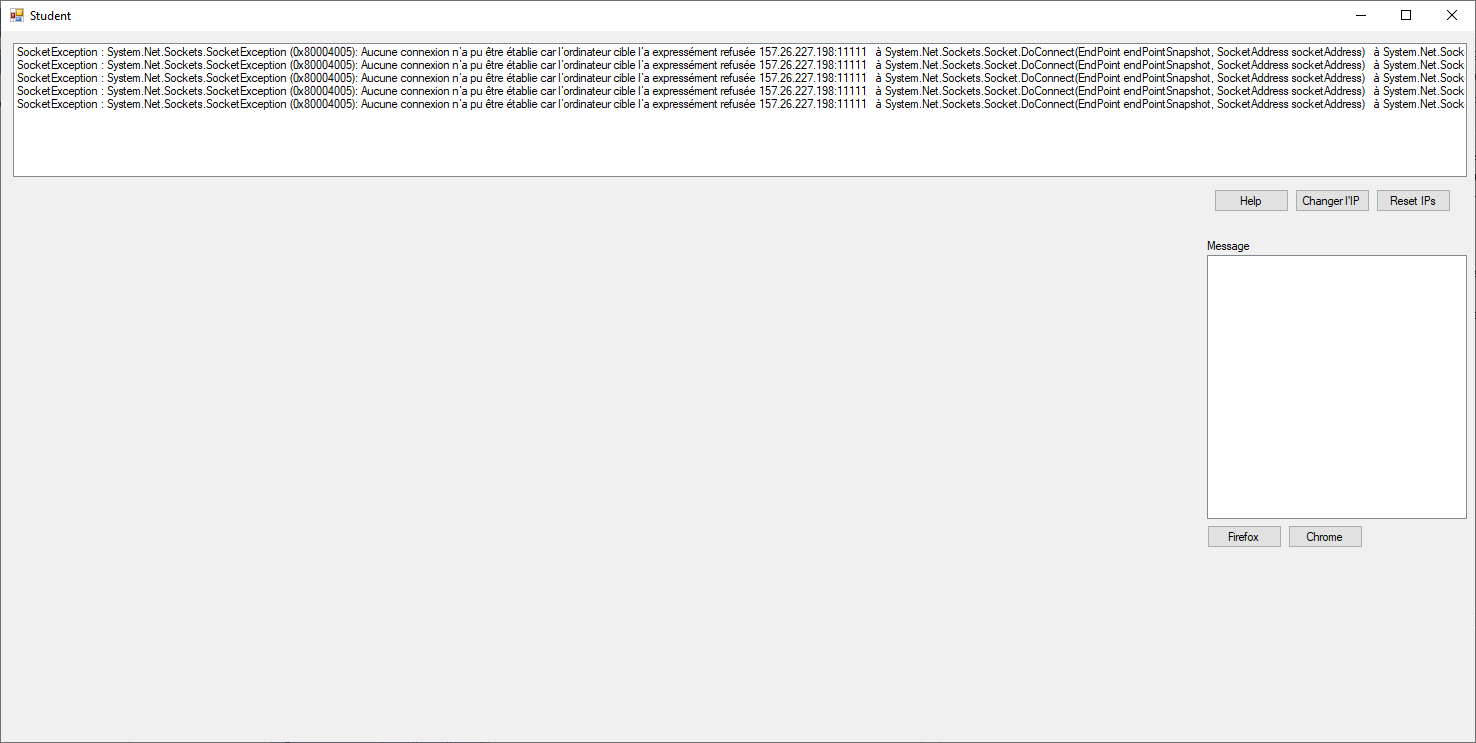
Le projet consiste à poursuivre ce développement de façon à disposer d’une alternative à Impero pour la surveillance des postes des élèves, la diffusion de démonstrations, la prise de contrôle pour effectuer une assistance à distance, le tout avec un logiciel léger et offrant des options non disponibles 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

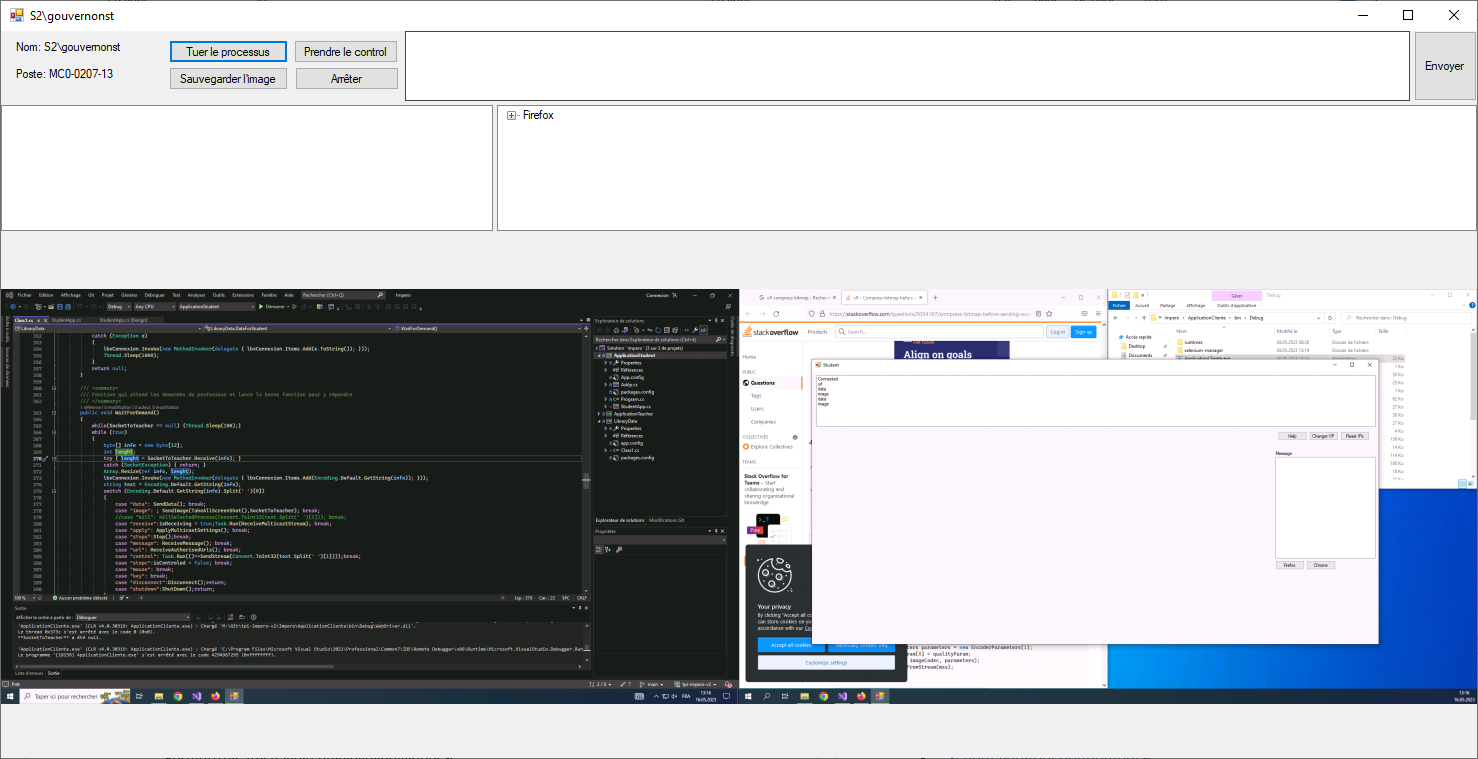
Description générée automatiquement

### Professeur

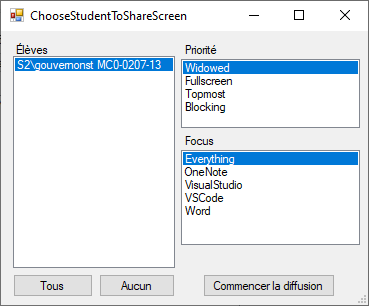
Principale



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 professeur.

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 élèves

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.

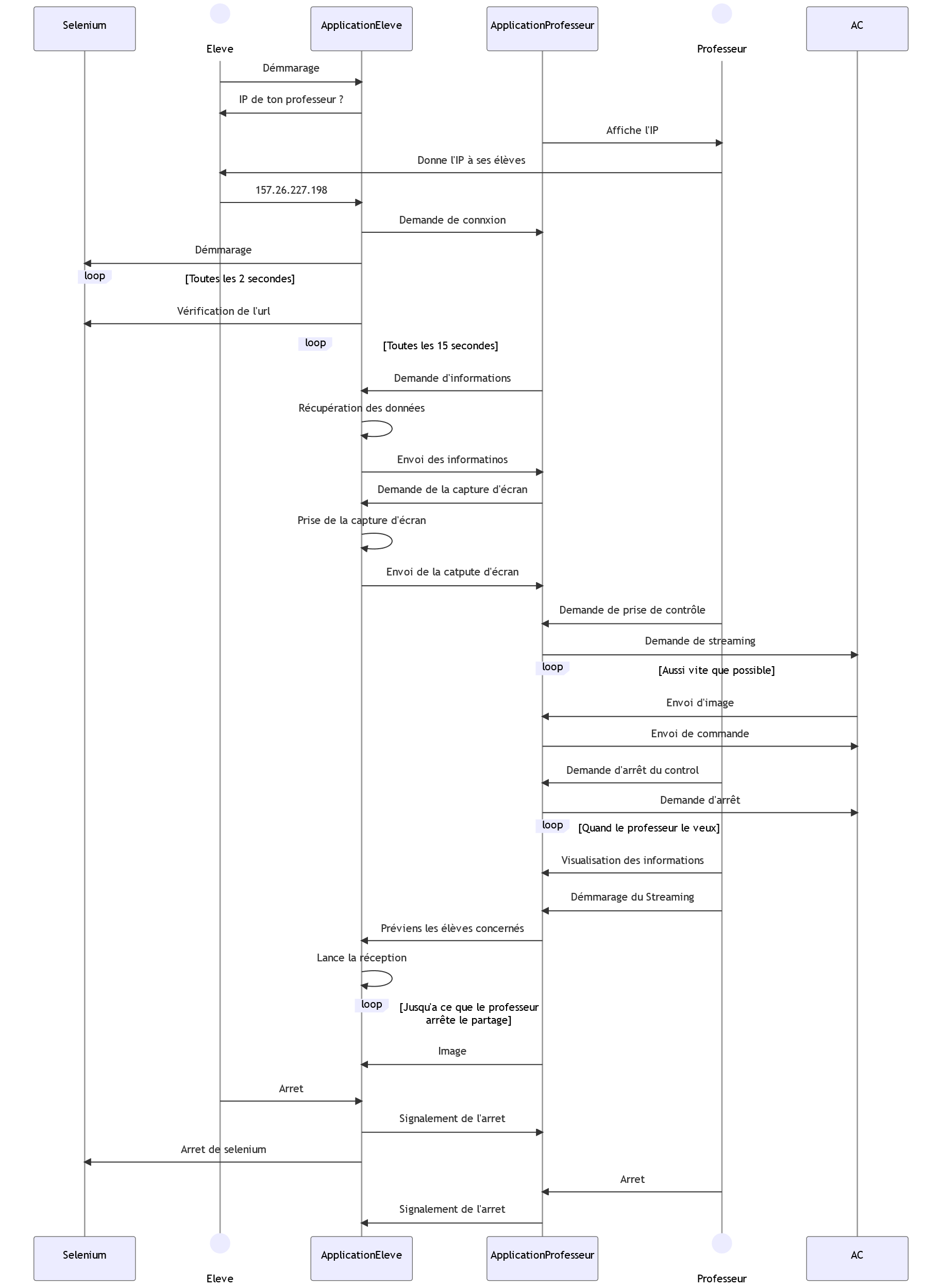
Envoi des données élèves au professeur.

Envoi des captures d’écran au professeur.

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

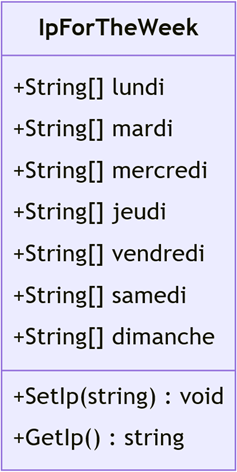
Partage d’écran du professeur en multicast.

## Schéma de fonctionnement

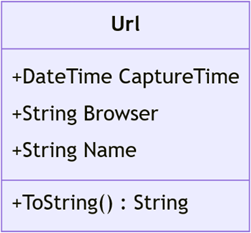


## Schéma de classe

IPForTheWeek



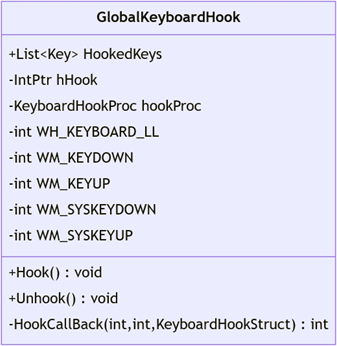
Url



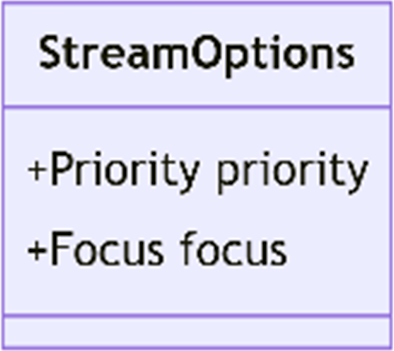
HistoriqueUrl



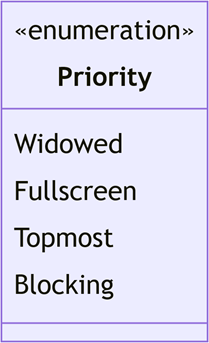
GlobalKeyboardHook



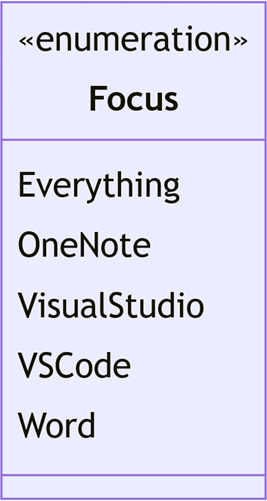
StreamOptions



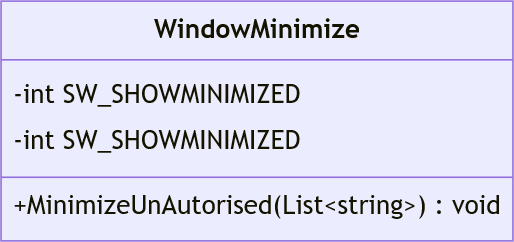
Priority



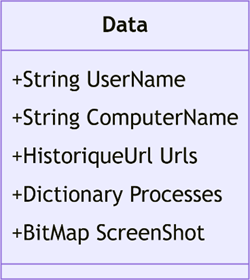
Focus



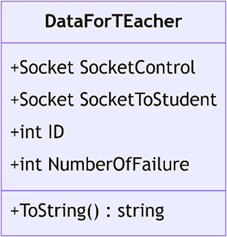
WindowMinimize



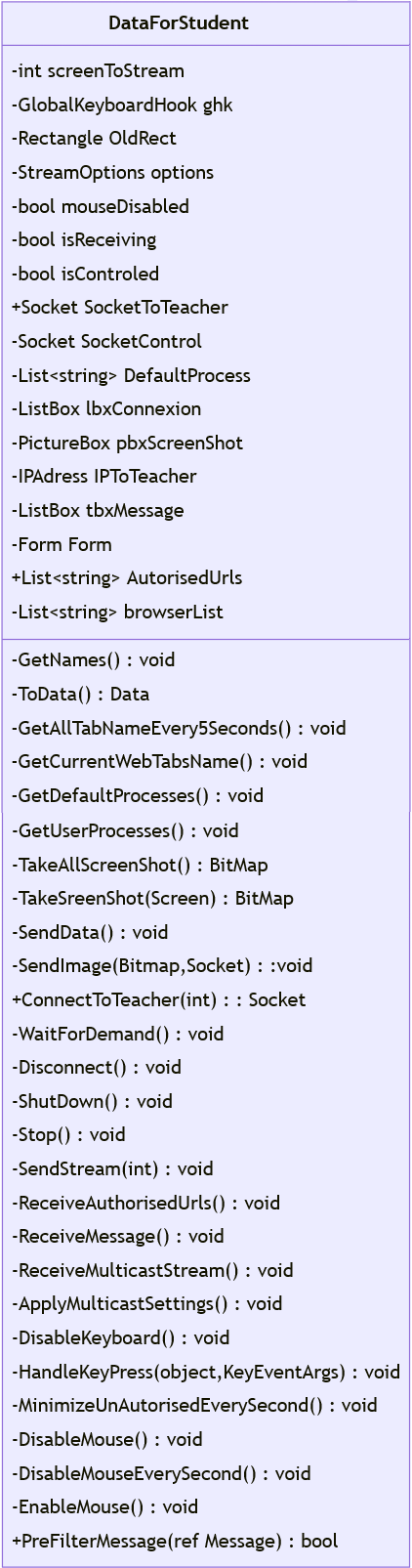
Data



DataForTeacher



DataForStudent



# Guide d’utilisateur

## Professeur

Au démarrage l’application essaye de déterminer quelle IP elle doit utiliser mais s’il y en a plusieurs vous aurez une fenêtre qui vous demanderas de choisir. Il faut que vous sélectionniez l’adresse qui est sur le même réseau que vos élèves.

Après le démarrage l’application demande et reçoit des données des élèves toutes les 15 secondes. Vous pouvez voir ces communications dans la ListBox en bas à droite de l’application.

Pour afficher les captures d’écran des élèves vous pouvez sélectionnez les élèves que vous voulez voir dans le TreeView en bas à gauche. Vous pouvez changer la taille des captures d’écran avec le slider en haut à gauche de l’interface.

En haut à gauche vous pouvez voir dans un TreeView tous les élèves connectés, en ouvrant les branches vous pourrez aussi voir les processus lancés par l’élève ainsi que les urls qu’il a visité sur chaque navigateur.

Le TreeView en bas à gauche vous permet aussi d’afficher une interface individuelle pour un élève. Pour cela il vous suffit de double cliquer sur un élève.

Maintenant que l’interface individuel est ouverte vous avez de nouvelles possibilités.

Arrêter l’application de l’élève.

Tuer un process sélectionné

Sauvegarder l’image actuel dans vos téléchargements

Prendre le contrôle de l’élève.

Le bouton tout en haut à gauche vous permet de partager votre écran.

Une nouvelle interface apparait et vous permet de choisir plusieurs options pour votre diffusion :

Les élèves à qui diffuser.

La priorité du Stream (pleine écran, fenêtré, au-dessus, blocage de l’élève).

Ainsi que les applications autorisées pendant la diffusion.

## 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.

Dans le cas ou des erreurs apparaissent dès l’ouverture dans le code il faut que vous désinstalliez puis réinstalliez le paquet « System.Text.Json » il ne semble pas toujours fonctionner correctement après un clonage sur git.

# 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 où l’application n’arrive pas à charger l’IP une interface permet à l’élève de rentrer l’IP de son professeur.

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

Puis deux tâches sont lancées en parallèles :

La connexion de l’élève au professeur.

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 :*

### *Trouver les processus lancés par l’utilisateur*

*Pour savoir quels processus ont été lancés par l’utilisateur, il suffit de comparer les processus actuels avec les processus par défaut trouvé au point précédent.*

### *Arrêter les processus lancés par l’utilisateur*

*Après avoir récupéré la liste des processus lancé par l’utilisateur il faut pouvoir arrêter le processus indésirable. Pour cela nous envoyons un dictionnaire avec le nom et l’id du processus à l’application maître. Elle peut ensuite quand elle le désire nous renvoyer l’id pour dire à l’application cliente de stopper ce processus.*

### *Capture d’écran des deux moniteurs*

*Prendre une capture d’écran de plusieurs moniteurs n’est pas facile car on ne peut pas capturer plusieurs écrans en même temps. Il faut donc faire plusieurs captures d’écran que l’on recompose ensuite en une seule image. Pour l’instant les captures d’écran ne sont pas comparées, en cas d’écran dupliqué ils seront les deux présents.*

### *Récupération des urls ouverts*

*Après de multiples essais la récupération de tous les onglets ouverts semble peut-être possible avec AutomationElement, mais cela serait long à apprendre et à faire pour tous les navigateurs. J’avais encore deux options : récupérer le nom de l’onglet actuel ou son url. Mais le code trouvé pour récupérer l’url était différent pour chaque navigateur et semblait ne fonctionner que sur certaines versions des navigateurs. N’importe quelle mise à jour pouvait donc demander une adaptation du code. Alors que La récupération du nom des onglets était indépendante de la version du navigateur et était identique pour tous les navigateurs. C’est pour cela que j’ai finalement choisi de ne récupérer que le nom de l’onglet ouvert actuellement.*

### 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 élève. Elle peut donc récupérer l’url actuel ainsi que d’interdire les url non autorisées en revenant en arrière.

### Réception de messages

L’application élève 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 élève 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

## Interface

## 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 prochaine 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.

Puis il va lancer deux tâches qui vont tourner continuellement en arrière-plan :

La première va permettre aux élèves de se connecter au professeur.

La deuxième va être une demande périodique des informations aux élèves.

## Fonctionnement

### *Spécialisation :*

### *Envoi de requêtes*

*L’application maître fait des demandes à toutes les applications clientes connectées, elle demande les données puis après les avoir reçues elle demande la capture d’écran quand elle la reçoit elle passe au prochain client. Elle boucle indéfiniment tant qu’elle a des applications clientes connectée.*

### *Affichage des données client*

*L’application maître doit permettre deux vues. Une avec tous les écrans des ordinateurs clients qui permet d’avoir une vue d’ensemble sur totalité des ordinateurs clients. Et une deuxième plus précise qui affiche toutes les informations sur un seul client, et permet de d’arrêter un processus chez ce client.*

### 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 peut 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 dès le TreeView Détails peut être activé ou non grâce au bouton « Activer/Désactiver » en haut à gauche de l’interface. Cela permet de ne pas afficher les processus ignorés ainsi que de faire apparaitre en rouge les processus et les urls alertés.

### 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 dès 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 élèves 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.

# Liaisons entre les deux applications

## Spécialisation :

## *Connexion*

*La connexion entre les deux applications se fait par une requête de l’application cliente à l’application maître. Pour cela l’application cliente doit connaitre l’adresse IP de l’application maître auquel elle doit se connecter. Cette connexion est utile pour créer un socket qui permet la connexion entre les deux applications à tout moment. C’est pour cela que l’application maître écoute pour des requêtes de connexion client en continu.*

Après des améliorations l’application cliente peut maintenant mémoriser les adresses IP des professeurs d’une semaine à l’autre. Ce qui permet à l’élève de ne rentre l’adresse IP de sont professeur qu’au début des cours. L’élève peut bien sur réinitialiser les adresses IP ou les changer.

## *Socket*

*Un socket permet d’établir une connexion permanente entre les applications, ils nous serviront à transmettre les données sous forme de tableau de byte entre l’application cliente et l’application maître. Au moment où l’une des applications s’arrête, elle doit prévenir l’autre pour qu’elle arrête son socket.*

## *Données*

*Les données ne peuvent pas être transférées tel quel, L’application cliente doit donc sérialiser les données cliente contenue dans ClientData avant de les transformer en un tableau de byte qui est la seule forme de donné qui peut être envoyé dans le socket. Á la réception, l’application maître devra retransformer le tableau de byte reçu en une chaine de caractères qu’il pourra ensuite désérialiser pour recréer la classe ClientData.*

L’application élève utilise désormais la classe DataForStudent qui hérite de data. Le Professeur utilise la classe DataForTeacher qui hérite elle aussi de la classe data. Et la classe data contient toutes les données qu’il faut transmettre. Je convertis donc la classe DataForStudent en Data pour l’envoyer sur le réseau puis c’est au professeur de créer une instance de DataForTeacher grâce à la class Data obtenue.

## *Images*

*L’utilisation de SendFile() pour envoyer une image semblait être une bonne idée mais finalement il était plus simple de transformer le fichier en un tableau de byte et de l’envoyer ainsi. Il suffit simplement à l’application maître de recréer une image à partir du tableau de byte.*

## *Flux d’image pour les élèves*

*Le flux d’image permet un partage vidéo de l’écran de l’application maître à toutes les applications clientes. Pour cela elle doit prendre une capture d’écran puis la séparer un plusieurs paquets pour pouvoir l’envoyer.*

*L’application cliente doit recevoir tous ces paquets puis recomposer une image avec puis l’afficher à l’utilisateur jusqu’à ce que la prochaine arrive.*

## Réception des demandes

La réception des demandes du professeur se fait dans une fonction dédiée à cela. Elle est constamment entrain d’écouter les demandes du professeur. Lorsqu’elle reçoit une demande elle exécute la bonne fonction pour répondre à la demande.

## Flux d’image pour le professeur

Lorsque le professeur veut prendre le contrôle d’un ordinateur c’est à l’application de l’élève de streamer son écran pour le professeur mais cette fois ce n’est pas en multicast ce qui nous permet d’utiliser le protocole TCP, je peux donc envoyer les images en un seul paquet de donnée. Mais je ne peux pas utiliser le même port sur l’application professeur cela risquerait de faire des collisions avec les communications.

# 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

# Problèmes et solutions

## TreeView customisé

Afficher des cases 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. Dans le cas où il n’y a qu’un seul IP elle est choisie automatiquement.

## 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 application 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 application en WPF

## WebView2 n’est pas complet

Pour avoir un navigateur intégré à mon application 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 effectué 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 ce qui permettait de ne plus perdre le delegate.

## Empêcher l’utilisation de certaines applications pendant le Stream

La première solution aurait été de fermer toutes les applications non autorisées mais on risquait la perte de données et de temps pour l’élève c’est pourquoi j’ai décidé de minimiser les fenêtres interdites durant le Stream, mais je ne pouvais pas empêcher les élèves de les agrandir à nouveau. C’est pourquoi je les minimise toutes les secondes.

# 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/>

Mettre à jour les binding List dans une application multithread : <https://stackoverflow.com/questions/4823481/bindinglist-not-updating-bound-listbox>

Hook pour capturer l’événement des touches du clavier : <https://www.codeproject.com/Articles/19004/A-Simple-C-Global-Low-Level-Keyboard-Hook>

Minimiser les autres fenêtres : <https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-showwindow>

## 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

Améliorer la stabilité des programmes,

Améliorer la qualité des Stream.

## Objectifs atteints

### Élève

### Professeur

## Conclusion