Documentation Outil de gestion d’appareils Android

1. Structure du projet

Le projet est un mono-repository c'est-à-dire que le front-end et le back-end sont dans le même fichier, néanmoins les fichiers du front-end et le back-end peuvent être séparés.

La racine de ce projet est une application Angular avec à l'intérieur un dossier "**/server**" contenant un projet node-js ainsi que tous les fichiers du back-end.

Pour réduire la taille du projet il est possible de fusionner les deux dossier **node_modules** en spécifiant dans l'un des fichiers package.json où est situé le dossier node_modules, nous ne l'avons pas faite pour que le front-end et le back-end restent facile à séparer

Grâce à cette structure mono-repository le front-end et le back-end peuvent plus facilement être lancés simultanément avec une seule commande:

Pour lancer le serveur en mode test utilisez la commande **'npm run test'**.
Pour lancer le serveur en mode production utilisez la commande **'npm run prod**.
(Utilisez ces commandes à partir de la racine du projet)

Ces commandes utilisent le module npm **concurrently**, un module qui permet d'exécuter plusieurs commandes à la fois:
https://www.npmjs.com/package/concurrently

Dans le dossier "**/server**", on retrouve :

- Le fichier **server.js** qui est exécuté en premier,
- Le fichier **routes.js** qui contient tous les endpoints du serveur,
- Le fichier **adb.js** qui contient toutes les fonctions que l'on utilise pour interagir avec les appareils.

2. Dépendances et installation

Le projet nécessite les outils suivants:

```
Android Debug Bridge version 1.0.39
NodeJS version 14.18.2
NPM version 6.14.15
Angular CLI version 13.1.1
```

Pour télécharger les outils nécessaires utilisez la commande **'sudo sh init.sh'** a la racine du projet.
Cette commande télécharge la version approprié de nodejs, npm, et de abd, puis démarre un serveur adb, et finalement installe les modules node requis par ce projet.

Si ces outils sont déjà installés et que vous voulez juste télécharger les modules node nécessaires, utilisez la commande **'npm run init'**.

3. Fonctionnement du serveur

Le serveur peut être lancé tout seul via la commande **'nodemon server/server.js'**.
(Utilisez cette commande à partir de la racine du projet)

Routes

Le serveur utilise **express** pour gérer les routes.

GET /devices

Cette route renvoie les appareils qui sont actuellement connectés au serveur via usb avec un array contenant les identificateurs des appareils.

POST /batterylevels

Cette route prends en paramètre une liste d'appareils et renvoie les niveaux de batteries des appareils avec un objet de la forme suivante:

{id1 : niveau_batterie, id2 : niveau_batterie, …}

POST /devicenames

Cette route prends en paramètre une liste d'appareils et renvoie les noms des appareils avec un objet de la forme suivante:

{id1 : nom1, id2 : nom2, …}

POST /shellcmd

Cette route prends en paramètre une liste d'appareils et une commande shell et renvoie la sortie de la commande sur les appareils avec un objet de la forme suivante:

{id1 : sortie, id2 : sortie, …}

POST /adbcmd

Cette route prends en paramètre une liste d'appareils et une commande adb renvoie la sortie de la commande sur les appareils avec un objet de la forme suivante:

{id1 : sortie, id2 : sortie, …}

POST /installpackage

Cette route route prends en paramètre une liste d'appareils et un fichier apk et renvoie "Success" si l'apk à bien été installé ou "" si l'apk est déjà installé avec un objet de la forme suivante:

*{id1 : "*Success"*, id2 : ""*, *…}*

ATTENTION: Si la version de l'apk ne convient pas à l'appareil, cela fait planter l'outil adb et en conséquence, le serveur. Pour l'instant le seul contournement est d'appliquer un timeout à la commande puis redonner la main.

POST /checkpackageinstalled

Cette route route prends en paramètre une liste d'appareils et un nom de package (com.example.appname) et renvoie un boolean qui indique si le package est installé sur l'application avec un objet de la forme suivante:

{id1 : true, id2 : false, …}

POST /installedpackages

Cette route prends en paramètre une liste d'appareils et renvoie la liste des packages installé sur les appareils passé en paramètre avec un objet de la forme suivante:

{id1 : [com.example.app1, com.example.app2, …] , id2 :[com.example.app3, com.example.app4, …], …}

POST /uninstallpackage

Cette route prends en paramètre une liste d'appareils et un nom de package et renvoie 'Success' si la désinstallation a bien marché, ou "" sinon avec un objet de la forme suivante:

{id1 : "Success", id2 : "", …}

POST /uninstallmuliplepackages

Cette route prends en paramètre une liste d'appareils et une liste de noms de package et renvoie pour chaque package 'Success' si la désinstallation a bien marché, ou "" sinon avec un objet de la forme suivante:

{package1: *{id1 : "Success", id2 : "", …}, package2: {id1 : "Success", id2 : "", …}, …}*

POST /pushfile

Cette route prends en paramètre une liste d'appareils et un fichier et renvoie une sortie console qui indique si le fichier a bien été transmis avec un objet de la forme suivante:

{id1 : "[100%] /sdcard/sample.txt sample.txt: 1 file pushed. 0.6 MB/s (2964 bytes in 0.004s)*", …}*

POST /pullfile

Cette route prends en paramètre une liste d'appareils et un chemin de fichier et renvoie le fichier s'il existe, sinon une erreur avec un objet de la forme suivante:

{id1: {success: true, data: 64bitData, filename: "id1_example.txt"}, id2: {success: false, error: "File /sdcard/example.txt does not exist on device id2"}, ...}

Les fichiers sont renommés avant d'être transmis par le serveur pour différencier de quel appareil vient le fichier.

POST /deletefile

Cette route prends en paramètre une liste d'appareils et un chemin de fichier et renvoie un message qui indique si le fichier a bien été supprimé avec un objet de la forme suivante:

{id1 : "/sdcard/example.txt was deleted successfully", id2 : "/sdcard/example.txt No such file or directory", id3: "Error sdcard/example.txt was not deleted", …}

POST /addwifi

Cette route prends en paramètre une liste d'appareils, un ssid, un mot de passe (peut être vide), un type de mot de passe (WPA, WEP, EAP ou None) et un nom d'utilisateur (pour les réseaux EAP) et retourne le réseau wi-fi sur lequel l'appareil est connecté avec un objet de la forme suivante:

{id1 : "eduroam", id2 : "none", …}

POST /disablewifi

Cette route prends en paramètre une liste d'appareils et retourne un booléen qui indique si la wi-fi est activé ou pas avec un objet de la forme suivante:

{id1 : false, id2 : false, …}

POST /enablewifi

Cette route prends en paramètre une liste d'appareils et retourne un booléen qui indique si la wi-fi est activé ou pas avec un objet de la forme suivante:

{id1 : true, id2 : true, …}

POST /recordscreen

Cette route prends en paramètre une liste d'appareils et un entier n et retourne un fichier mp4 d'un enregistrement de n seconds de l'écran de l'appareil avec un objet de la forme suivante:

{id1_recording_1639580035179.mp4: *64bitData, …}*

Pour distinguer les fichiers, ils sont nommés avec l'identificateur de l'appareil ainsi qu'un timestamp.

POST /screencapture

Cette route prends en paramètre une liste d'appareils et retourne un fichier png d'un capture d'écran de l'appareil avec un objet de la forme suivante:

{id1_screencap_1639580186515.png: *64bitData, …}*

Pour distinguer les fichiers, ils sont nommés avec l'identificateur de l'appareil ainsi qu'un timestamp.

POST /getwificonnection

Cette route prends en paramètre une liste d'appareils et retourne le réseau wi-fi sur lequel l'appareil est connecté avec un objet de la forme suivante:

{id1 : "eduraom", id2 : "none", …}

Transfert de fichiers

Pour transférer les fichiers depuis le serveur jusqu'au front-end, les fichiers sont encodés sous format 64 bits puis envoyés en tant que chaîne.

Le transfert de fichier du client jusqu'au serveur est limité à 100 Mo mais cela peut être modifié dans le fichier **server/server.js**

Adb

Pour exécuter des commandes adb, le serveur utilise le module **child_process**, qui permet de lancer des sous processus similaire à **popen**

Appareils

La solution est conçue spécifiquement pour les Samsung SM-T580, et il se peut que certaines fonctionnalités ne marchent pas sur d'autres appareils, notamment la gestion de wi-fi.

Les appareils doivent avoir le mode développeur activé avec l'option 'Débogage USB' activé.
La première fois que les appareils sont connectés (via USB) au serveur web, il faudrait sur chaque appareil valider un boîte de dialogue pour que l'appareil puisse 'faire confiance' au serveur.

Gestion de wi-fi

Pour gérer les connexions wi-fi des appareils, on utilise une application Android capable de recevoir des commandes adb et de rajouter des connexions et modifier les connexions existantes.
https://github.com/steinwurf/adb-join-wifi

A noter:

- Nous avons modifié cette application pour ne pas qu'il reste ouvert quand son traitement est terminé et pour que l'application puisse ajouter des réseaux EAP.

- Seulement les réseaux ajoutés par cette application peuvent être modifiés par cette application.

- Avant d'envoyer des commandes adb à cette application la solution vérifie d'abord si elle est installée et sinon l'install, c'est pour cette raison que le fichier apk apparaît dans le dossier /server.

- L'application est cachée, et ne peut être utilisée que depuis adb.

- L’application ne peut pas supprimer de réseaux mais en donnant le mauvais mot de passe, l'appareil ne pourrait plus se connecter dessus.

- Les sources de cette application se trouvent dans le fichier adb-add-wifi.zip