POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

Département de génie informatique et génie logiciel

INF3995 : Projet de conception d'un système informatique

Exigences techniques
Conception d'un système audio pour café Internet

Complément au document de demande de proposition no. H2019-INF3995

Version 1.1

Anas Balboul
Benjamin Heinen
Jérôme Collin, ing., M. Sc. A,

Janvier 2019

Table des matières

But du document

Le présent document complète l'appel d'offres H2019-INF3995 en précisant les exigences techniques dont les soumissionnaires devront tenir compte pour présenter des propositions conformes.

Les propositions devront détailler et démontrer comment ces exigences pourront être rencontrées. Elles devront également servir de point de départ à l'élaboration du calendrier des diverses tâches à réaliser.

Aperçu du produit demandé

Le système à concevoir repose sur une communication client/serveur. Le client sera principalement la tablette mobile et le système à FPGA jouera le rôle de serveur. Une application client PC devra aussi interagir avec le serveur pour gérer le système.

Le rôle principal du serveur est de faire le décodage MP3 et d'acheminer les échantillons de son décodés vers la sortie audio standard (connecteur 3.5mm). Le serveur doit aussi répondre aux différentes requêtes d'appareils Android reliés au système Wi-Fi du réseau local, dont certaines pour permettre la supervision des activités par le personnel.

Normalement, le serveur fait jouer les chansons dans l'ordre qu'il les reçoit des usagers ordinaires, tout simplement. La simplicité d'utilisation doit être au cœur du système puisqu'on désire avant tout en faire un système qui gagnera en popularité.

L'application Android viendra en deux modes : un pour les superviseurs et un pour les clients du café-bistro. L'interface sera relativement simple dans les deux cas.

Dans le mode pour les clients, il est possible de savoir la liste des chansons qui seront prochainement jouées et de soumettre des fichiers MP3 soi-même. Aucun contrôle n'est possible sur les chansons que l'on n'a pas soumises soi-même. Pour celles que l'on a soumises, on peut les retirer de la liste. Il y a une limite au nombre de chansons qu'on peut avoir en attente sur le serveur par usager. Le but étant de pouvoir laisser la chance à tous les clients de pouvoir soumettre des chansons.

Le mode superviseur peut gérer toutes les chansons en attente (retirer de la file, changer de position dans la file, etc.) Le superviseur contrôle aussi le volume du son en sortie. Il peut également bloquer certaines adresses IP de clients que l'on désire exclure du système. Normalement, un gérant ou un responsable du bistro durant les heures d'ouverture est un superviseur. Il ne peut soumettre lui-même des chansons.

L'administrateur crée des comptes superviseur. Il a aussi accès à un module de statistiques qui donne une certaine idée de la popularité du système. Il peut aussi avoir accès à toutes les activités (*logs*) du serveur. L'administrateur ne peut soumettre lui-même des chansons. L'administrateur est le propriétaire du bistro ou une autre personne à qui il veut déléguer cette responsabilité. L'administrateur interagit avec le serveur via une application client sur PC (Windows ou Linux) complémentaire au système. Il n'y a qu'un seul administrateur possible dans le système.

Système

Cette section décrit le système envisagé. Il s'agit ici d'un logiciel s'exécutant sur un système embarqué pour ce qui est du serveur et de deux applications client, une sur Android et l'autre sur PC.

Matériel du serveur :

Tel qu'énoncé précédemment, les périphériques disponibles sur le système sont ceux de la carte de développement Zedboard. Celle-ci est munie d'un FPGA/SoC Zynq XC7Z020 de Xilinx, lequel contiendra la logique nécessaire à l'exécution de l'appareil (bus, processeurs, pilotes, intermédiaires logiciel/matériel, etc.) On suppose que l'équipe de développement est déjà familière avec l'environnement de développement de Xilinx et de Linux. L'environnement de développement recommandé est la suite Xilinx Vivado/SDK 2018.1 et une distribution Arch Linux avec interface graphique Xfce.

Le système doit être muni, au minimum, des ressources suivantes :

- deux processeurs ARM Cortex-A9 proprement configurés pour le contexte;
- un accès à une mémoire vive externe de 512 MiB;
- un accès à une mémoire SD externe;
- un accès à l'interface réseau;
- un pilote pour contrôler l'interface HDMI (vidéo);
- un pilote pour contrôler le son en sortie (connecteur audio 3.5mm);
- un pilote pour contrôler un UART (pour une console en mode administrateur);

Logiciel sur serveur

- Développer le serveur en C/C++;
- Mentionner les versions de librairies importantes utilisées;
- Fournir les fichiers de configuration;
- Prévoir qu'il faudra fournir les commandes d'installation de ces logiciels;
- Faire valider auprès du promoteur tout logiciel supplémentaire utilisé dans le produit final livré;
- Placer le serveur dans l'environnement de *systemd* pour qu'il puisse démarrer automatiquement à l'amorçage (*boot*) du système Linux.

Logiciel de l'application Android

- Code développé en Java ou en Kotlin;
- Tablette mobile avec Android 6 ou plus;
- Utilisation d'Android Studio pour le développement;
- Bien identifier les librairies importantes utilisées;
- Faire valider auprès du promoteur tout logiciel supplémentaire utilisé dans le produit final livré.

Logiciel de l'application PC

- Application avec logiciel compilé ou interprété nativement sous Linux ou Windows;
- Aucun langage de programmation imposé;
- L'interface graphique ne peut pas être effectuée avec des technologies web, mais avec une librairie prévue pour utilisation native (sans l'aide de fureteur Internet) et autonome (standalone);

Le serveur sur FPGA

Le serveur devra tourner sur Linux Arch. Cependant, il ne devrait pas dépendre de détails techniques très spécifiques à cette plate-forme et devrait, dans la mesure du possible, profiter d'interfaces assez génériques et standards de Linux pour opérer convenablement.

La gestion du son sous Linux prend différentes formes. Il sera de la responsabilité du soumissionnaire de préciser de quelle façon il envisage la solution qui lui semble la plus appropriée. Il n'y a peut-être pas de mauvaise solution, mais il pourrait y en avoir de meilleures que d'autres du point de vue de la performance.

Le serveur sera obligatoirement écrit en C/C++ et probablement avec l'aide de la bibliothèque Boost. Pour le décodage, la librairie libmad doit être utilisée directement. On ne pourra pas utiliser une autre forme de code disponible sur Internet pour compléter le décodage.

Le serveur devra décoder les en-têtes ID3 associés au format des fichiers MP3. Plusieurs bibliothèques permettent d'y arriver. De cet en-tête, le serveur pourra tirer les renseignements importants sur une chanson: titre, artiste et durée. Les versions 1 et 2 de ID3 doivent toutes deux être supportées correctement.

Un ajustement doit être fait pour que la sortie audio reproduise la chanson au bon rythme et selon les valeurs prévues dans l'en-tête des trames MP3. Ainsi, la fréquence des échantillons de son (*sample frequency*) doit être réglée correctement, tout comme l'aspect stéréo/mono de l'encodage.

Le serveur doit répondre aux requêtes des tablettes et de l'application PC. L'interface REST sera décrite plus bas. Quelques librairies supplémentaires peuvent être requises pour arriver à les gérer de façon simple et efficace, tout en conservant un système relativement fluide. Le nombre de chansons dans la file est limité. Dans le code, l'associer à une constante. 10 pourrait être une bonne valeur, mais on peut aller à moins pour des raisons de tests du système.

Il devra être possible de démarrer automatiquement le serveur avec *systemd* au démarrage (*boot*) de Linux. Cette étape ne devrait pas être trop complexe en suivant ce qui est mentionné ici.

L'application Android – Mode utilisateur ordinaire

L'application Android pour le client du café-bistro qui veut utiliser le système est assez simple. Il doit y avoir une option permettant de balayer l'ensemble de l'appareil pour trouver les fichiers MP3 disponibles en vue d'établir une liste à présenter à l'utilisateur. Cette option doit toujours pouvoir être relancée pour mettre à jour la liste. Cette opération peut toujours être effectuée même si l'application n'est pas connectée au serveur.

L'application n'utilise pas de compte pour se connecter au serveur. On assume que tous les gens qui se sont connectés au service Wi-Fi du lieu peuvent l'utiliser sans compte. On veut favoriser la participation sans que l'utilisation devienne trop compliquée. Tout de même, on veut avoir une option qui permette de se connecter au serveur (via un bouton par exemple). Dès que la connexion est activée, l'application reçoit du serveur la liste des chansons déjà dans la file et qui seront bientôt jouées. Périodiquement, l'application peut redemander au serveur cette liste pour conserver l'application à jour par rapport à l'état du serveur.

Une autre option de l'interface sera pour accéder à sa propre liste de fichiers MP3 (précédemment établie). Avec cette liste, il faudra prévoir dans l'interface un moyen de préciser l'envoi d'un ou de quelques fichiers MP3 au système. Le serveur devra les entrer dans sa file de chansons à jouer. Un maximum de 5 fichiers d'un usager peut se retrouver au même moment dans la file du serveur.

Les chansons que l'usager a soumises au système devraient apparaître d'une couleur différente par rapport à celles soumises par les autres utilisateurs pour que l'usager sache lesquelles sont les siennes. L'usager peut retirer volontairement ses propres chansons de la file du serveur, mais pas celles des autres utilisateurs du système. Il faudra prévoir un mécanisme intuitif de retrait volontaire des chansons de l'utilisateur.

En temps normal, l'adresse IP du serveur serait intégrée à l'application sans possibilité de modification. Dans le processus de développement, comme on utilisera peut-être quelques serveurs différents pour des fins de tests, il serait important d'avoir une interface permettant de préciser l'adresse IP du serveur (ou au moins le dernier 8 bits).

Comme l'aspect sécuritaire est quelque peu relâché pour favoriser la facilité d'utilisation, il serait bien que l'application prenne une couleur jaune, avec une écriture noire, les couleurs du café-bistro Élévation. Ces traits distinctifs permettront au moins au personnel de service circulant de table en table de voir discrètement si les gens utilisent l'application avec facilité ou semblent être confus devant l'interface. L'endroit n'étant pas très grand, la vérification devrait être possible assez facilement.

Enfin, il faudra prévoir dans l'application un moyen d'ouvrir une portion de l'interface (ou fragment) donnant les paroles de la chanson lorsque c'est possible. Vous pouvez vous inspirer de ce script pour y arriver, même s'il est en python. On ne peut assurer que la procédure fonctionnera pour toutes les chansons, mais l'application devra faire de son mieux pour y arriver lorsque les données sont disponibles.

L'application Android – Mode superviseur

Il n'y a pas d'application séparée pour le mode superviseur. Les fonctionnalités sont côte à côte par rapport à celles pour l'utilisateur ordinaire. Tout de même, il faudra prévoir une façon d'accéder à ce mode depuis l'interface principale de l'application à développer.

Contrairement au mode usager ordinaire, on devra fournir un code d'utilisateur et un mot de passe pour accéder à un compte superviseur via une connexion sécurisée. Les comptes superviseurs sont créés par l'administrateur dans une application séparée.

Tout comme pour le mode usager ordinaire, le mode superviseur reçoit la file actuelle des chansons soumises au serveur lors de la connexion et cette requête doit être refaite périodiquement pour maintenir l'application à jour. Pour les chansons listées, le superviseur voit en plus de quelle adresse IP la chanson a été soumise ainsi que l'adresse MAC associée.

Le superviseur peut retirer n'importe quelle chanson de la file et même réordonner des chansons de la file. Il peut gérer le volume du système ou le mettre en sourdine (mute) quelques instants. Le superviseur ne peut pas soumettre de chansons lui-même.

Le superviseur peut aussi bloquer un usager par son adresse IP ou MAC. Il faudra donc maintenir une liste noire et la gérer (ajouter ou retirer des éléments de la liste). Cette liste noire est globale et partagée parmi tous les superviseurs du système.

Le mode administrateur et son application

Il ne doit y avoir qu'un seul compte administrateur dans le système développé. Il doit être accessible uniquement à travers une application sur PC soit en Linux, soit en Windows. L'application doit offrir en premier lieu l'occasion à l'administrateur d'entrer son identificateur et son mot de passe. Par la suite, l'administrateur doit avoir plein contrôle sur le système à travers une connexion sécurisée.

L'administrateur doit avoir un moyen de créer et supprimer des comptes de superviseurs pour ceux qui font la gestion du restaurant durant les heures d'ouverture. Le nombre de comptes de superviseurs restera tout de même fort limité en pratique. Il ne devrait pas nécessiter le support d'une base de données importante, par exemple. Ceci ne limite toutefois pas le contracteur dans le choix de sa solution proposée s'il désire avoir recourt à une base de données.

Un autre aspect important du mode administrateur est la capacité de suivre les activités du serveur. Le serveur devra tenir informer l'application de bureau (*desktop*) de l'administrateur des évènements importants suivants, sous forme de « logs » en format texte :

- Création ou suppression d'un compte de superviseur;
- Émission d'un nouvel identificateur d'usager ordinaire;
- Soumission d'une nouvelle chanson (ajout à la file);
- Début du décodage d'une nouvelle chanson par le système;
- Retrait d'une chanson (par un superviseur ou un usager régulier);
- Modification à l'ordre de passage de chansons;
- Connexion ou déconnexion d'un compte de superviseur;
- Blocage ou déblocage d'un usager par un superviseur;
- Modifications au volume du son (incluant la sourdine - *mute*).

Le mode administrateur se terminera par un module de statistiques qui affichera le nombre de chansons jouées depuis le début de la journée et la durée moyenne de celles-ci (minutes et secondes). On voudra aussi savoir le nombre d'utilisateurs différents ayant utilisé le système et le nombre de chansons retirées volontairement (par les usagers ou le superviseur).

Les communications entre cette application sur PC et le serveur doit se faire avec chiffrement. Le choix du langage de programmation utilisé pour la création de l'application est laissé à la discrétion du soumissionnaire. De même, une interface présentant les informations de manière ergonomique, simple et efficace est attendue. Cependant, aucune directive stricte sur sa conception n'est imposée. La créativité du soumissionnaire est sollicitée pour cet aspect du système. Aucune technologie Web n'est permise pour la réalisation de l'interface graphique. Une possibilité est d'utiliser Qt, mais il y a d'autres options possibles.

L'interface entre le serveur et le client (utilisateur ordinaire)

L'interface entre le mode usager ordinaire et le serveur est décrite ici. Les deux entités communiqueront au moyen d'une interface REST et du protocole HTTP dont voici les détails. Les communications se font sur le port 80 du serveur.

Requêtes HTTP

```
GET /usager/identification
```

Cette requête GET est la première dans la séquence des opérations puisqu'elle permet à l'usager d'obtenir un jeton pour utiliser le système. Un fichier JSON doit accompagner la requête.

Le fichier JSON aura la structure suivante:

```
{
   "ip": [0-255]:[0-255]:[0-255]:[0-255],
                            // adresse IP d'où provient la requête
   "MAC": [0-F][0-F]:[0-F][0-F]:[0-F][0-F]:[0-F][0-F]:[0-F][0-F]:[0-F][0-F],
                           // adresse MAC d'où provient la requête
   "nom": string           // nom ou pseudonyme de l'utilisateur
                           // s'il souhaite s'identifier.  Peut-être nulle ("").

}
```

Le serveur retournera le JSON suivant si le client est autorisé à recevoir l'information.

```
{
   "identificateur": integer, // Choisi par le serveur et unique à cet usager.
                              // Valide pour la journée seulement
                              // zéro si l'utilisateur ne peut pas se connecter
   "message": string  // message à afficher à l'usager de l'application
                      // exemple: Bienvenue au café-bistro Élévation!
                      // exemple 2: Désolé, vous n'êtes plus autorisé à utiliser le
                      //            système du café-bisto Élévation...
}
```

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : OK
400 : mauvaise requête (erreur dans le JSON)
403 : accès refusé
500 : erreur serveur (service non disponible)

```
GET /usager/file/<Id>
```

Cette requête GET est la seconde dans la séquence des opérations puisqu'elle permet à l'usager d'obtenir la liste des chansons en attente d'être jouées.

Le serveur retournera le JSON suivant si le client est autorisé à recevoir l'information.

```
{
   "chansons": [
       { "titre": string, "artiste": string, "duree": mm:ss,     // forme générale
         "proposeePar": string, "proprietaire": bool, "no": integer },
       // note: proprietaire indique que cet usager a lui-même soumis cette
       // chanson (true) ou si elle a été soumise par un autre usager (false)
       // note 2: no est un numéro unique de chanson dans le système durant
       // la journée.  Il devrait croître linéairement durant la journée.
       { "titre": "Hey Jude", "artiste": "Beatles", "duree": "7:05",  // exemple
         "proposeePar": "Claude", "proprietaire": False, "no": 25 },
       ...
   ]
}
```

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : OK
403 : accès refusé
500 : erreur serveur (service non disponible)

`POST /usager/chanson/<Id>`

Cette requête permet à l'usager (ayant son identificateur *Id*) de soumettre une chanson. Il n'y a pas de JSON à soumettre, car les renseignements sur la chanson se trouvent à l'intérieur de l'en-tête ID3 du fichier MP3 lui-même et le serveur sait de qui elle provient par l'identificateur unique.

Cependant, le fichier MP3 lui-même suit la requête par un mécanisme quelconque. Une possibilité est qu'il soit encodé en base64 dans le corps de la présente requête. Il devra être décodé par le serveur éventuellement si ce mécanisme est retenu. Par contre, ouvrir une connexion séparée au serveur par un autre canal est aussi possible.

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : ok
403 : accès refusé
413 : chanson trop longue, file pleine sur le serveur ou limite de chansons dans la file pour l'usager atteinte
415 : le fichier soumis n'est pas un MP3 ou n'a pas d'en-tête ID3
500 : erreur serveur (service non disponible)

`DELETE /usager/chanson/<Id>/<no>`

Cette requête permet à l'utilisateur de supprimer une chanson qu'il a déjà lui-même soumise. Le numéro de chanson *no* est celui global du système.

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la

transaction.

Statuts HTTP: 200 : ok
403 : accès refusé
405 : refusé (la chanson n'appartient pas à l'usager ou mauvaise chanson)

L'interface entre le serveur et le client (mode superviseur)

L'interface entre le mode superviseur et le serveur est décrite ici. Les deux entités communiqueront au moyen d'une interface REST et protocole HTTP sécurisé dont voici les détails. Tous les échanges se feront avec une connexion SSL sur le port 443.

Requêtes HTTPS

`POST /superviseur/login`

Pour se connecter au système en tant que superviseur.

Le fichier JSON à envoyer au serveur aura la structure suivante:

```
{
   "usager": string,          // le champ est le nom du compte du superviseur
   "mot_de_passe": string
}
```

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : ok
400 : mauvaise requête (erreur dans le JSON)
403 : non autorisé

`POST /superviseur/<usager>/logout`

Pour se déconnecter, évidemment. Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : ok
401 : utilisateur non authentifié

`POST /superviseur/<usager>/changement_mdp`

Pour le changement de mot de passe d'un superviseur.

Le structure JSON à envoyer au serveur aura la structure suivante:

```
{
   "ancien":  string,
   "nouveau": string
}
```

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : ok

400 : mauvaise requête (mauvais nouveau mot de passe ou
erreur dans le JSON)
401 : utilisateur non authentifié

`GET /superviseur/<usager>/file`

Cette requête GET est la première dans la séquence des opérations puisqu'elle permet à un superviseur *usager* d'obtenir la liste des chansons en attente d'être jouées. Le serveur retournera le JSON suivant qui est semblable à celui pour l'usager ordinaire, mais avec plus de détails sur la personne qui a soumis la chanson et son appareil mobile.

```
{
   "chansons": [
       { "titre": string, "artiste": string, "duree": mm:ss,     // forme générale
         "ip": [0-255]:[0-255]:[0-255]:[0-255],
         "MAC": [0-F][0-F]:[0-F][0-F]:[0-F][0-F]:[0-F][0-F]:[0-F][0-F]:[0-F][0-F],
         "Id": integer, "proposeePar": string, "no": integer },

       { "titre": "Hey Jude", "artiste": "Beatles", "duree": "7:05",  // exemple
         "ip": "192.168.0.67", "MAC": "F4:4D:40:6B:3A:23", "Id": 892311241,
         "proposeePar": "Claude", "proprietaire": False, "no": 25 },
       ...
   ]
}
```

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : OK
401 : utilisateur non authentifié
500 : erreur serveur (service non disponible)

`DELETE /superviseur/<usager>/chanson/<no>`

Cette requête permet au superviseur de supprimer une chanson. Le numéro de chanson *no* est celui global du système.

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : ok
401 : utilisateur non authentifié
405 : refusé (la chanson n'appartient pas à l'usager ou mauvaise chanson)

`POST /superviseur/<usager>/position`

Cette requête permet au superviseur de positionner une chanson dans la file du système d'un ancien endroit vers une nouveau. Le numéro de la position 1 dans la file est la prochaine chanson à jouer. Le format JSON à utiliser est le suivant:

```
{
   "ancien":  integer,    // exemple: 3
   "nouveau": integer     // exemple: 1
}
```

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : ok
400 : mauvaise requête (erreur dans le JSON)
401 : utilisateur non authentifié
409 : la chanson n'est pas dans la liste ou ne peut être placée à l'endroit demandé

`GET /superviseur/<usager>/volume`

Cette requête permet de savoir à quelle valeur le volume audio est réglé présentement en pourcentage en sortie sur la fiche de 3.5mm du serveur. L'état de la sourdine est aussi fourni. Le serveur retourne le résultat dans un format JSON:

```
{
   "volume": integer,   // entre zéro et cent.  Exemple: 65
   "sourdine" : bool    // sourdine activée (True) ou non (False)
}
```

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : ok
401 : utilisateur non authentifié

`POST /superviseur/<usager>/volume/augmenter/<pc>`

Cette requête permet d'augmenter le volume audio de *pc* pourcent en sortie sur la fiche de 3.5mm du serveur.

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : ok
401 : utilisateur non authentifié

`POST /superviseur/<usager>/volume/diminuer/<pc>`

Cette requête permet de diminuer le volume audio de *pc* pourcent en sortie sur la fiche de 3.5mm du serveur.

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : ok
401 : utilisateur non authentifié

`POST /superviseur/<usager>/volume/sourdine/activer`

Cette requête permet de mettre en sourdine (*mute*) la sortie sur la fiche de 3.5mm du serveur.

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : ok
401 : utilisateur non authentifié

`POST /superviseur/<usager>/volume/sourdine/desactiver`

Cette requête permet désactiver la mise en sourdine (*mute*) la sortie sur la fiche de 3.5mm du serveur (le son se fait entendre de nouveau).

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : ok
401 : utilisateur non authentifié

`POST /superviseur/<usager>/bloquer`

Cette requête permet au superviseur de bloquer l'adresse IP (et MAC, en fait pour être vraiment précis car c'est elle qui est constante pour un appareil) d'un usager ordinaire. Un JSON avec les informations suivantes doit accompagner la requête :

```
{
   "ip": [0-255]:[0-255]:[0-255]:[0-255],
                            // moins utile qu'on pense étant donné qu'elle est
                            // obtenue par le DHCP du café-bistro et donc changeante
   "MAC": [0-F][0-F]:[0-F][0-F]:[0-F][0-F]:[0-F][0-F]:[0-F][0-F]:[0-F][0-F],
                           // vraiment ce qui doit être bloqué, car unique
                           // donné, car plus facile à retenir que l'adresse MAC.
   "nom": string           // Peut donc être utile pour un repérage rapide par
                           // le superviseur si le nom de l'usager est aussi présent
}
```

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : ok
400 : mauvaise requête (erreur dans le JSON)
401 : utilisateur non authentifié

`POST /superviseur/<usager>/debloquer`

Cette requête fait l'inverse de la précédente. Un JSON avec les informations suivantes doit accompagner la requête :

```
{
   "MAC": [0-F][0-F]:[0-F][0-F]:[0-F][0-F]:[0-F][0-F]:[0-F][0-F]:[0-F][0-F],
                          // vraiment ce qui doit être débloqué, car unique
}
```

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : ok
400 : mauvaise requête (erreur dans le JSON)
401 : utilisateur non authentifié

`GET /superviseur/<usager>/listenoire`

Cette requête permet d'obtenir la liste des adresses MAC bloquées. Le nom de la personne ayant soumis (si connu) et la dernière adresse IP associées sont aussi retournés si l'information peut aider l'administrateur. Le serveur retournera un vecteur des adresses au format JSON suivant :

```
{
   "bloques": [
       { "ip": [0-255]:[0-255]:[0-255]:[0-255],         // forme générale
         "MAC": [0-F][0-F]:[0-F][0-F]:[0-F][0-F]:[0-F][0-F]:[0-F][0-F]:[0-F][0-F],
         "nom": string },

       // exemple:
       { "ip": "192.168.0.88", "MAC": "4B:3E:58:99:79:B2", "nom": "Claude" },
       ...
   ]

}
```

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : ok
401 : utilisateur non authentifié

L'interface entre le serveur et le client (mode administrateur)

L'interface entre le mode administrateur et le serveur est décrite ici. Les deux entités communiqueront au moyen d'une interface REST et protocole HTTP sécurisé dont voici les détails. Tous les accès au serveur du mode superviseur seront avec une connexion SSL sur le port 443.

Requêtes HTTPS

`POST /admin/login`

Pour se connecter au système en tant qu'administrateur.

Le format JSON à envoyer au serveur aura la structure suivante:

```
{
   "usager": string,          // le champ est obligatoirement "admin"
   "mot_de_passe": string
}
```

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : ok
400 : mauvaise requête (erreur dans le JSON)
403 : non autorisé

`POST /admin/logout`

Pour se déconnecter, évidemment. Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : ok
401 : utilisateur non authentifié

`POST /admin/changement_mdp`

Il n'y a qu'un seul compte. L'identificateur de ce compte est obligatoirement « admin » et seul ce compte pourra voir son mot de passe être changé.

Le JSON à envoyer au serveur aura la structure suivante:

```
{
   "ancien":  string,
   "nouveau": string
}
```

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : ok
400 : mauvaise requête (mauvais nouveau mot de passe ou erreur dans le JSON)
401 : utilisateur non authentifié

`GET /admin/statistiques`

Cette requête est utilisée pour obtenir les statistiques depuis l'ouverture du café-bistro tôt le matin pour le déjeuner. Le bistro-café ne veut pas se lancer dans des chiffres à n'en plus finir, car les dirigeants pensent qu'ils sauront bien assez rapidement si le service est apprécié, ou non, et probablement en ayant des commentaires de gens directement plutôt que par la compilation de résultats détaillés. Un JSON avec les informations de base est retourné par le serveur :

```
{
   "chansons": integer,  // nombre de chansons soumises par l'ensemble des usagers
   "utilisateurs": integer, // nombre d'utilisateurs différents ayant soumis
                            // des chansons
   "elemines" : integer,    // nombre de chansons retirées par les superviseurs
   "temps" : mm:ss          // durée moyenne des chansons soumises
}
```

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : ok
401 : utilisateur non authentifié

`POST /admin/logs`

La requête pour obtenir les informations du serveur sur les activités importantes. Les informations fournies par le serveur doivent être numérotées. L'application mentionne le numéro du dernier renseignement reçu dans le JSON envoyé pour éviter de recevoir de nouveau de l'information déjà reçue. Si ce nombre est zéro, ceci indique que l'application n'a encore rien reçu jusqu'à maintenant. Dans ce cas, on peut retourner uniquement les 20 derniers éléments d'information disponibles sur le serveur.

Le fichier JSON à envoyer au serveur aura la structure suivante:

```
{
   "dernier": integer   // dernier renseignement numéroté reçu
}
```

Le serveur retournera un JSON donnant les messages d'information numérotés :

```
{
   "information": [
      {
        "no": integer,   // numéro du renseignement
        "info": string   // message
      },
```

```
     . . .
      {
        "no": integer,
        "info": string
      }
   ]
}
```

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : ok
400 : mauvaise requête (erreur dans le JSON envoyé)
401 : utilisateur non authentifié

`POST /admin/creationcompte`

Pour créer un compte superviseur avec le nom d'usager et le mot de passe spécifiés dans le JSON qui accompagne la requête.

Le fichier JSON à envoyer au serveur aura la structure suivante:

```
{
   "usager": string,          // nom du compte superviseur
   "mot_de_passe": string     // mot de passe initial
}
```

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : ok
400 : mauvaise requête (erreur dans le JSON)
403 : non autorisé

`POST /admin/suppressioncompte`

Pour supprimer un compte superviseur ayant le nom d'usager spécifié dans le JSON qui accompagne la requête.

Le fichier JSON à envoyer au serveur aura la structure suivante:

```
{
   "usager": string           // nom du compte superviseur
}
```

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction.

Statuts HTTP: 200 : ok
400 : mauvaise requête (erreur dans le JSON ou nom d'usage inexistant)
403 : non autorisé

Contenu des livrables

Le café-bistro Élévation demande deux phases de développement, chacune aboutissant à un livrable fonctionnel selon les critères explicités dans cette section des exigences techniques.

Livrable 1

Pour le **livrable 1**, évalué le jeudi 28 mars 2019, le café-bistro Élévation Inc. s'attend aux fonctionnalités suivantes :

- Serveur :
  - tout le décodage du son, mais pas les ajustements de volume (ni la sourdine) ;
  - la file, mais sans possibilités de retraits volontaire de chansons ;
- Application Android :
  - tout le mode usager ordinaire au complet sauf le retrait des chansons ;
- Application sur PC :
  - les « logs » du serveur ;
  - sans login et sans connexion sécurisée nécessairement ;

Livrable 2

Au **livrable 2**, évalué le jeudi 11 avril 2018, vous devez terminer le projet, ce qui implique, par rapport au livrable 1 :

- Serveur :
  - la gestion de la liste noire ;
  - les modifications de la file (retraits, déplacements de chansons) ;
  - ajustements du volume et de la sourdine ;
  - Amorçage avec systemd dans ArchLinux ;
- Application :
  - le mode superviseur au complet ;
- Application sur PC :
  - le module de statistiques ;
  - la création/suppression des comptes de superviseurs ;
  - Authentification et connexion sécurisée ;

Note : Les grilles d'évaluation utilisées par le promoteur seront disponibles aux entrepreneurs avant les évaluations des livrables.