POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

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. A2018-INF3995

Version 1.1

Charles Hosson
Patrick Richer St-Onge
Jérôme Collin, ing., M. Sc. A,

Septembre 2018

Table des matières

But du document

Le présent document complète l'appel d'offres A2018-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 la tablette mobile et le système à FPGA jouera le rôle de serveur. 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.

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 l'administrateur 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 d'en soumettre 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. Le but étant de pouvoir laisser la chance à tous les clients de pouvoir soumettre des chansons.

Le mode administrateur peut gérer toutes les chansons en attente (retirer de la liste, changer de position dans la file, etc.) L'administrateur contrôle aussi le volume du son en sortie. L'administrateur peut bloquer certaines adresses IP de clients. L'administrateur a aussi accès à un mode statistique qui donne une certaine idée de la popularité du système. L'administrateur ne peut soumettre lui-même des chansons.

Système

Cette section décrit le système envisagé. Il s'agit ici d'un logiciel s'exécutant sur un système embarqué muni d'un FPGA.

Matériel

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

Les contraintes logicielles :

- 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;
- Fournir un conteneur Linux du serveur dans sa version finale.

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.

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 est généralement utilisée, même dans le populaire logiciel VLC. On la recommande. Par contre, MP32PCM est aussi une option possible (et bien documentée par un livre), de même que mpg123. Il y aura peut-être un peu de profilage à faire pour s'assurer de la performance de l'ensemble du travail de décodage à effectuer.

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.

Le serveur doit répondre aux requêtes des tablettes. L'interface REST sera décrite plus bas. L'utilisation de quelques librairies supplémentaires peut-être requis 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.

Un mécanisme permettant de suivre les activités du serveur devra être mis en place. On peut vouloir utiliser le traditionnel système Unix syslog ou établir un équivalent en écrivant dans un fichier texte directement. Il sera important d'enregistrer les évènements suivants :

- 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 le superviseur ou un usager régulier);
- Modification à l'ordre de passage de chansons;
- Connexion et déconnexion du compte du superviseur;
- Blocage ou déblocage d'un usager par l'administrateur;
- Modifications au volume du son (incluant la sourdine - *mute*).

Pour un déploiement rapide du système, la solution finale du serveur devra être livrée dans un « container » Linux (Docker, probablement) en plus du code source complet.

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'envoie 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.

L'application Android – Mode superviseur (administrateur)

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 le seul compte administrateur possible du système. Tous les aspects de sécurité et de connexion sont détaillés dans une section séparée, plus bas.

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. L'administrateur 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).

Enfin, le mode superviseur se terminera par un petit module de statistique 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).

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.

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 et doit être encodé en base64. Il devra être décodé par le serveur éventuellement.

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 dont voici les détails. À noter que la prochaine section présentera ce qui est attendu de la connexion sécurisée au système.

Requêtes HTTPS

`GET /superviseur/file`

Cette requête GET est la première dans la séquence des opérations puisqu'elle permet à l'administrateur 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/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/inversion`

Cette requête permet au superviseur d'inverser l'ordre de passage de deux chansons dans la file du système. Les numéros de chanson sont ceux globaux du système.

```
{
   "une": integer,    // exemple: 31
   "autre": integer   // exemple: 33
}
```

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 : l'une ou l'autre (ou les deux) chansons ne sont plus dans la liste en attente

`GET /superviseur/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/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/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/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/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

`GET /superviseur/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 le superviseur
   "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 /superviseur/bloquer`

Cette requête permet au superviseur de bloquer l'adresse IP (et MAC, en fait) 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
   "nom": string           // donné, car plus facile à retenir que l'adresse MAC.
                           // Peut donc être utile pour un repérage rapide par
                           // le 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)
401 : utilisateur non authentifié

`POST /superviseur/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/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

Connexion au serveur et sécurité

Les accès par un usager ordinaire ne seront pas sécurisés. Tout sera en clair. De toute façon, il y a peu à sécuriser, car il n'y a pas réellement de données confidentielles. Le mode superviseur devra cependant être sécurisé. Tous les accès au serveur du mode superviseur seront avec une connexion SSL sur le port 443.

Il reste également à préciser les interfaces pour se connecter/déconnecter au système et la façon de changer le mot de passe de l'administrateur.

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 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 /superviseur/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/changement_mdp`

Pour l'instant, 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 fichier 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

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 mardi 13 novembre 2018, 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 volontaires ou d'inversion de chansons ;
- Application : tout le mode usager ordinaire au complet sauf le retrait des chansons;

Livrable 2

Au **livrable 2**, évalué le jeudi 29 novembre 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, inversions) ;
    - ajustements du volume et de la sourdine
- Application : le mode superviseur au complet ;

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