POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

Département de génie informatique et génie logiciel

INF3995 : Projet de conception d'un système informatique

Exigences techniques pour la conception d'un système de prédiction de données sur l'usage du BIXI

Complément au document de demande de proposition no. A2020-INF3995

Version 1.0

Anes Belfodil
Adem Aber Aouni
Jérôme Collin, ing., M. Sc. A,

Septembre 2020

Table des matières

Table des matières 2
But du document 3
Aperçu du produit demandé 4
Bases technologiques du système 5
  Logiciel sur serveur web central: 5
  Logiciel des engins de données: 5
  Déploiement du serveur web central et des engins de données: 5
  Logiciel de l'application Android : 5
  Logiciel de l'application PC: 5
Le serveur web central 7
Les engins de données 8
L'application Android 9
L'application PC 10
Les conteneurs Docker et Docker Compose 11
Les forêts aléatoires et données de base 12
Logs et messages 13
L'interface du serveur web 14
L'interface des engins de données 17
Contenu des livrables 21
  Prototype 21
  Livrable intermédiaire 21
  Livrable final 21

But du document

Le présent document complète l'appel d'offres A2020-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 qui communiquera avec un serveur web central écrit en C++ via une interface de type REST. Une application client PC devra aussi interagir avec le serveur web pour gérer le système. Un certain nombre d'autres engins de données effectueront de l'apprentissage et seront écrits en langage Python. Ces engins seront aussi des serveurs web dans un certain sens puisqu'ils répondront également à certaines requêtes spécifiques de la tablette, en plus de gérer les données et de procéder à l'apprentissage.

L'usager se connecte directement au système, sans compte. Cependant, il est invité à remplir certaines informations (nom, courriel, etc.) pour signifier qu'il a utilisé le système et s'il a un intérêt pour BIXI. Le serveur central collecte ces informations dans sa base de données. Le serveur central contrôle aussi les engins de données pour s'assurer qu'ils sont opérationnels.

Les engins de données font réellement l'analyse des données et l'apprentissage, parfois pour des opérations simples (comme retourner un graphique), mais aussi pour les plus complexes (apprentissage). Ils sont ceux qui répondront directement à la tablette pour des demandes de renseignements précis selon ce que chacun de ces engins a comme responsabilité. Les engins travailleront à partir du même ensemble de données qui seront les fichiers originaux des données du Bixi du site kaggle.com. En fait, les engins pourraient être regroupés en un seul, mais pour des raisons de séparation du travail et le fait de concevoir un système à plusieurs conteneurs indépendants justifie ce choix.

L'ensemble du serveur central et des divers engins de données doivent être regroupés dans un réseau de conteneurs Linux Docker et l'ensemble doit pouvoir être démarré avec l'utilitaire Docker Compose. Un gestionnaire de trafic web devra procéder à la répartition des requêtes selon les responsabilités de chacun. Cette conteneurisation devrait assurer une certaine forme de portabilité à l'ensemble.

Une application PC indépendante devrait monitorer le système et surtout récupérer les divers messages (logs) des engins et du serveur central. Il présentera également une vue des gens ayant fourni leurs renseignements au départ.

Le reste du document décrit les particularités supplémentaires de chaque composante du système ainsi qu'une description complète de l'interface REST entre les diverses composantes de service et les clients.

Bases technologiques du système

Cette section décrit le système envisagé du point de vue de la programmation et des diverses composantes principales.

Logiciel sur serveur web central:

- Développer le serveur en C++;
- Mentionner les versions de librairies importantes utilisées;
- Faire valider auprès du promoteur tout logiciel supplémentaire utilisé dans le produit final livré;

Logiciel des engins de données:

- Développer ces programmes en Python 3.6 et plus;
- Prévoir les librairies pour l'intelligence artificielle et lister les libraires utilisées;
- S'assurer de la flexibilité et de la robustesse de ces engins;
- Les messages (logs) produits durant le déroulement sont importants pour le promoteur;

Déploiement du serveur web central et des engins de données:

- Le serveur web central sera dans son propre conteneur Docker;
- De la même façon, chaque engin de données devra être placé un conteneur Docker;
- Un proxy-inverse (reverse proxy) Traefik devra être utilisé dans la solution pour le déploiement;
- Le regroupement des conteneurs devra se faire avec Docker Compose avec une version précise de votre choix, mais bien documentée;

Logiciel de l'application Android :

- Code développé en Java ou en Kotlin ;
- Tablette mobile avec Android 7 ou plus, mais fonctionnant obligatoirement sur une Galaxy Tab A avec Android 9;
- Utilisation d'Android Studio pour le développement;
- Bien identifier les librairies importantes utilisées;
- Faire valider auprès du promoteur tout logiciel supplémentaire ou librairie peu usuelle 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 réalisée avec des technologies web, mais avec une librairie prévue pour utilisation native (sans l’aide de fureteur Internet) et autonome (standalone);

Le serveur web central

Le serveur web devra tourner sous Linux. Cependant, il ne devrait pas dépendre de détails techniques très spécifiques à une distribution précise et devrait, dans la mesure du possible, profiter d'interfaces assez génériques et standards de Linux pour opérer convenablement.

Le serveur sera obligatoirement écrit en C/C++. On pourra avoir recourt à une bibliothèque comme Boost au besoin qui offre beaucoup de composantes de base. D'autres librairies à code source ouvert seront aussi requises pour des besoins précis (gestion du format JSON, etc.) Comme il s'agit d'un serveur avec interface REST vers les clients, il devra gérer les requêtes des clients avec une librairie de votre choix. Réalistement, ce choix ira probablement vers une des trois options suivantes :

- Restbed ;
- Libnavajo ;
- Pistache IO ;

Le serveur doit répondre aux requêtes des tablettes et de l'application PC. L'interface REST sera décrite plus bas.

Le serveur web interagit également avec les engins de données pour s'assurer qu'ils sont en service et maintenir l'application Android informée de la santé globale du système. Une solution spécifique et non précisée ici est attendue du soumissionnaire sur ce point.

Le serveur pourrait avoir besoin d'une petite base de données pour y stocker l'information sur les informations fournies par l'usager au départ. SQLite ou MySQL (ou ses quelques variantes) devraient suffire, mais rien n'empêche l'entrepreneur d'y aller avec une autre solution qui pourrait mieux convenir si la raison est bien justifiée.

Les engins de données

Les engins de données seront responsables de traiter les données, de faire l'apprentissage et de fournir les résultats demandés (prédictions, graphiques, données). Ils devront aussi répondre à l'application Android et à l'Application PC selon le type d'information que chacun est responsable de fournir. Les engins de données doivent aussi répondre au serveur web central, mais le protocole d'échange doit être élaboré par le fournisseur et n'est pas spécifié dans les requis techniques.

Bien entendu, ce projet se situe dans le domaine des systèmes distribués. Quelques lectures secondaires peuvent être utiles pour comprendre plus en détail certains concepts, même si on n'entre pas nécessairement dans des fonctionnements très complexes dans ce projet.

Il y a 3 engins de données :

- Le premier s'occupe des stations : recherche par mots clefs, fournir le code d'une station et pouvoir positionner sur la carte la station.
- Le deuxième voit à fournir des données assez brutes des fichiers. Il ne fait donc rien en rapport avec l'intelligence artificielle. Cet engin doit en plus de certains graphiques.
- Le dernier engin est celui qui fait réellement des prédictions et doit utiliser l'intelligence artificielle pour y arriver.

Dans l'interface REST plus loin dans ce document, il deviendra très clair quel engin doit servir une requête précise. Ceci n'empêche pas les 3 engins d'utiliser le même ensemble de fichiers de données pour faire leur travail. D'ailleurs, on voudra justement que les engins commencent leur travail à partir des fichiers originaux du site kaggle.com concernant les températures et les données du BIXI.

Les engins doivent tous être codés en Python 3 (on vous recommande une version de 3.6 en montant). Ils doivent être robustes. Ils feront nécessairement appel à plusieurs librairies et il faudra prévoir une bonne méthode de gestion d'installation de celles-ci. De toute façon, la création d'un Docker pour chacun d'eux l'imposera indirectement. Ils doivent donc manipuler des données, mais rester réactifs aux demandes du serveur central (vérifications d'activité), de la tablette (requêtes) et de l'application PC (rapporter les *logs*).

L'application Android

L'application Android se veut assez simple, puisqu'il suffit d'entrer ses informations d'usager et de consulter les données du système. Il y a tout de même des graphiques à afficher et le promoteur s'attend à une présentation des données dans une agréable à consulter. L'application doit être intuitive et facile à consulter. On devra accorder une grande importance à bien identifier les titres, les axes des graphiques, les unités de mesure des données présentées et des informations complémentaires appropriées pour éviter la confusion de l'usager qui peut être peu habitué à consulter des statistiques. Cette remarque est d'autant plus importante que l'analyse est le fruit d'avancées complexes dans le domaine de l'intelligence artificielle.

Les couleurs de l'organisme BIXI étant le rouge et le blanc, l'application Android devra prendre ces couleurs (consulter le site de l'organisme pour prendre la couleur exacte). On peut aussi ajouter un peu de noir au besoin, bien entendu. Les autres couleurs devront être utilisées avec parcimonie, essentiellement pour aider à la navigation et présenter des options à l'usager. Les recommandations du Material Design de Google se répandent de plus en plus et deviennent la voie à suivre.

Les requêtes HTTP REST plus loin dans le document permettront au soumissionnaire d'analyser le type d'information à afficher et la manière dont cette même information peut être obtenue du serveur central et des engins de données. Il appartient au soumissionnaire de proposer une solution bien adaptée à la visualisation des données et une navigation fluide et intuitive à travers les différentes fonctionnalités de l'application. L'usager n'a pas à savoir d'où proviennent les données (serveur web central, engins de données numéro X). Il ne voit que des informations à propos de BIXI.

À noter que l'application devra faire une requête régulièrement pour tester si le système n'a pas perdu contact avec l'un ou l'autre des engins de données. Cette procédure est rendue nécessaire puisque le système peut prendre un peu de temps pour effectuer certains traitements (particulièrement pour les prédictions) et retourner certains résultats. Il est toujours rassurant pour l'usager de savoir que le système est toujours en état de marche même s'il est en attente de ces résultats.

La très grande majorité des applications Android suivent de plus en plus l'architecture MV-VM, une variante du modèle classique MVC (modèle-vue-contrôleur). MVVM permet cependant au modèle de modifier la vue indirectement quand on lui demande, autrement dit quand les données changent (le «binding» des données et de la vue). Android recommande une façon de faire avec Jetpack. Avec Android, il faut surtout très bien comprendre les activités et les fragments. Les services sont plus intuitifs à assimiler.

Une tablette Samsung Galaxy Tab A de 10.1 pouces d'écran tournant Android 9.0 vous sera prêté pour la durée du projet. L'évaluation du projet se fera avec cette tablette. Cependant, rien n'empêche le développement sur des versions antérieures d'Android (souvent plus stables et mieux documentées) pour des raisons de commodités. Le consortium reste conscient que l'entrepreneur pourrait déjà avoir à sa disposition des appareils mobiles Android auxquels il est habitué et qu'il peut souhaiter utiliser à sa guise durant le développement.

L'application PC

L'application PC est comme l'application Android, une application qui est de type client dans le système. Il est prévu qu'elle ne soit employée que par une seule personne qui aura donc un rôle d'administrateur ou d'administratrice.

Il faut pouvoir fournir un mot de passe pour démarrer les opérations et se connecter au système après le démarrage de l'application PC. Par la suite, l'application a essentiellement 3 fonctionnalités bien distinctes :

- Pouvoir changer le mot de passe de l'unique compte administrateur du système;
- Pouvoir présenter les résultats du sondage avec les données récoltées jusqu'à maintenant;
- Pouvoir voir les messages (*logs*) de chacun des 3 engins de données de manière séparées pour chacun;

On pourra avoir un bouton pour redemander la totalité des données du sondage puisque celles-ci peuvent s'ajouter à tout moment. Pour les messages, il faudra procéder de façon plus incrémentale et faire sorte que le déroulement des engins puisse être suivi en continu sans avoir à redemander l'ensemble. Ainsi, au fur et à mesure qu'un engin produit des données, on devrait pouvoir les voir défiler automatiquement et de façon fluide dans l'application PC. Il serait préférable d'avoir 4 portions d'écran (ou 4 tabulateurs) pour voir les résultats du sondage et les messages de chacun des 3 engins. Par contre, le fournisseur a le loisir de soumettre une proposition originale pour faciliter la lecture des informations de ces 4 sources distinctes.

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. Il est possible que l'application fonctionne uniquement sur Windows ou uniquement sur Linux. Elle peut aussi être portable et convenir aux deux systèmes.

On mentionne ici les messages ou *logs* mais il faudra prendre en compte ce qui est mentionné dans la section sur les messages et réaliser que le mécanisme devra prendre en charge autant des informations textuelles que graphiques ce qui en fera une partie plus complexe qui si seulement des chaînes de caractères devaient être prises en charge.

Les conteneurs Docker et Docker Compose

S'affranchir des particularités d'un système d'exploitation et avoir une solution facilement déployable est une préoccupation importante des systèmes informatiques modernes. Une méthode s'est grandement répandue ces dernières années: Docker. Après avoir lu l'article court et simple sur Wikipédia, il est recommandé de suivre ce tutoriel sur le sujet.

Il faudra avoir un conteneur pour le serveur web central et un pour chacun des engins de données. De plus, ils devront être regroupés dans un même réseau virtuel Docker à l'aide de l'outil de contrôle Docker Compose dont le fonctionnement est aussi expliqué dans le tutoriel sur Docker. Attention toutefois, car les différentes versions de Docker Compose entraînent des variations de syntaxe importante parfois incompatibles les unes avec les autres. Assurez-vous d'utiliser une version fixe et bien précise pour toute l'équipe. Bien chercher à vérifier quelle version de l'outil a été utilisée lorsque vous verrez les exemples de fichier de configuration sur internet.

L'utilitaire Traefik devra orchestrer l'ensemble du trafic web en jouant le rôle de mandataire inverse. Plus spécifiquement, Traefik devra :

- Gérer les certificats de sécurité autosignés pour le site;
- Rediriger les requêtes REST au bon conteneur (serveur web central ou engins de données);
- Rediriger automatiquement le port 80 vers le port 443;
- Offrir une interface web de monitorage Traefik sur le port 8080;
- Condamner l'accès aux ressources que l'usager ne devrait pas pouvoir consulter;

Le Docker Compose devra aussi bien gérer les volumes menant aux répertoires contenant les données manipulées par les engins de données.

Les fichiers de configuration de Docker et Docker Compose font partie intégrante des requis exigés par le requérant et doivent être remis par le fournisseur.

Les forêts aléatoires et données de base

L'intelligence artificielle est un domaine vaste et complexe. Une façon plus facile d'aborder le sujet est à partir des forêts aléatoires, qui fonctionnent à la base à partir d'arbres de décisions. C'est encore mieux de le faire à partir de deux articles complémentaires (le premier, le deuxième) accompagnés d'un exemple de code en Python directement. Ceci permet aussi de suivre les étapes importantes et générales pour aborder un problème en intelligence artificielle. Refaire cet exemple avec le code fourni devrait être une étape importante pour la compréhension de la préparation des données en intelligence artificielle. La lecture de l'article avec l'exécution du code en parallèle devrait fournir un bon tour d'horizon des étapes importantes.

L'exemple des deux articles vise la prédiction des températures à partir de données existantes. La démarche générale est présentée puisqu'elle introduit à la préparation des données ce qui peut représenter 80% du code alors que lancer les algorithmes qui font réellement le travail avec certaines librairies peut se faire en quelques lignes.

Il pourrait manquer quelques notions complémentaires pour mieux comprendre certains aspects. Ces vidéos sur youTube peuvent aider. Les quelques premières sont intéressantes pour avoir une meilleure idée de l'apprentissage automatique. Aussi, le « bagging » est introduit aux sections 9.1 à 9.3.

Le présent projet vise le même genre de démarche, mais à partir des données d'utilisation du système BIXI, disponibles sur kaggle.com. On remarque une bonne quantité de données de 2014 à 2017. Si on va plus loin, on peut même voir un exemple complet de traitement IA avec la température et l'utilisation de BIXI. C'est une démarche très similaire qu'il faudra suivre, mais en utilisant les forêts aléatoires, et non les moteurs plus complexes utilisés dans cet exemple. Néanmoins, avec cet exemple complet et les deux articles précédemment mentionnés, il devrait être possible d'atteindre l'objectif avec le système. Les données sur les températures à pour la même période peuvent être trouvées aussi sur le site kaggle.com.

Logs et messages

L'administrateur du système doit avoir un moyen de vérifier que les engins de données opèrent correctement et que les données sont manipulées correctement. Il n'y aura pas de format précis imposé pour fournir les informations jugées pertinentes. Une évaluation du livrable intermédiaire permettra au promoteur de mesurer la progression de l'avancement des travaux de cette partie du produit demandé et de préciser des correctifs au fournisseur. Les améliorations pourront concerner l'esthétique dans la présentation de l'information ou encore la quantité et l'exactitude des messages.

Néanmoins, un bon point de départ à considérer pour l'élaboration de cette partie de la solution est l'article numéro 2 sur les arbres aléatoires. Quelques principes directeurs peuvent être dégagés de ce qui est attendu :

- Transmettre les commandes Python importantes dans les *logs*;
- Inclure des portions d'en-têtes de fichiers, de tableaux ou de vecteurs au moment jugé important pour permettre au lecteur d'avoir une idée de leur contenu;
- Présenter des graphiques de résultats importants dans le traitement, au besoin. Lorsque ce même graphique est retourné à l'application Android, on peut éviter de le répéter dans les *logs* inutilement;
- Dans le cas des prédictions par intelligence artificielle, il faudra au minimum fournir le graphique de l'arbre aléatoire obtenu (ou une portion de celui-ci s'il est trop imposant);
- Ajouter des messages significatifs sur le déroulement, mais sans nécessairement expliquer les concepts du traitement non plus;

Bien entendu, cette portion du système sera importante pour le promoteur dans l'évaluation globale de l'exactitude des opérations du système, mais elle devrait également permettre au soumissionnaire de plus facilement vérifier et tester sa solution en cours de développement.

L'interface du serveur web

L'interface du serveur web central et les applications Android et PC. Les entités communiqueront au moyen d'une interface REST et du protocole HTTPS dont voici les détails. Les communications se font sur le port 443 du serveur.

Requêtes HTTPS

`POST /usager/login`

Utilisé pour vérifier le mot de passe du compte **«** admin » utilisé par l'application PC. Cette requête devra d'accompagner du champ HTTP `Authorization` avec le mot de passe actuel. En effet, comme l'application PC est la seule à récupérer des données sensibles du serveur, elle devra fournir une sécurité minimale, soit celle de fournir un nom de compte (toujours le même – admin) et un mot de passe selon le mécanisme d'authentification de base du protocole HTTP. Ce mécanisme d'authentification n'a pas à être mis en place entre l'application Android et le serveur web central puisque n'importe qui peut consulter les informations.

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

Statuts HTTP: 200 : ok
403 : utilisateur non autorisé

`PUT /usager/motdepasse`

Utilisé pour changer le mot de passe du compte **«** admin » utilisé par l'application PC pour recueillir les données du sondage. Cette requête devra d'accompagner du champ HTTP `Authorization` avec le mot de passe actuel avant son changement.

Le JSON à envoyer au serveur aura la structure suivante contenant le nouveau mot de passe:

```
{
   "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

`PUT /sondage`

Pour envoyer les données du sondage depuis la tablette vers le serveur. La précédente entrée avec le même courriel est remplacée par celle-ci si elle existait. Le mécanisme d'authentification de base ne doit pas être utilisé avec cette requête puisqu'elle provient de la tablette.

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

```
{
   "courriel": string   // permet d'identifier une entrée unique dans le système
   "prenon": string,
   "nom": string,
   "age": integer,
   "interet": true|false // si l'usager souhaite être contacté par BIXI dans le futur
}
```

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)

`GET /sondage`

Pour que l'application PC puisse recueillir les données du sondage du serveur web central. Le mécanisme d'authentification de base doit être utilisé avec cette requête. En cas de succès, le serveur central retournera une liste complète des structures de données d'usagers. Notez que la liste peut être vide :

```
{
   {
      "courriel": "jsmith@exemple.com",
      "prenon": "Joe"
      "nom": "Smith"
      "age": 48,
      "interet": true
   },
   . . .
}
```

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

Statuts HTTP: 200 : ok
400 : mauvaise requête

```
GET /status
```

Cette requête devra être faite au serveur périodiquement (par exemple, au 10 secondes) par l'application Android de manière à s'assurer de l'intégrité du système de données du côté serveur. Le serveur web central doit servir cette requête après s'être lui-même assuré que les engins de données sont aussi opérationnels. Le JSON suivant sera retourné par le serveur web central en cas d'erreur :

```
{
   "message":  string, // pour donner plus d'information à l'usager sur Android
}
```

En cas d'erreur veut dire la perte de contact d'un engin de données dans le système. On assume ici que le serveur web, lui, ne peut pas tomber hors service. Autrement, le système ne peut pas faire la moindre opération. Comment le serveur web central s'assure que les engins de données sont tous actifs est laissé à l'initiative du fournisseur. Le mécanisme d'authentification de base ne doit pas être utilisé avec cette requête.

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

Statuts HTTP: 200 : ok
500 : erreur d'un engin de données

L'interface des engins de données

L'interface entre les applications Android et PC d'une part et les 3 engins de données d'autre part. Ils communiqueront au moyen d'une interface REST et du protocole HTTP sécurisé sur le port 443 dont voici les détails.

Requêtes HTTPS

`POST /station/recherche`

Pour effectuer une recherche de nom de stations dans la liste avec une chaîne de caractères. Cette requête devra être prise en charge par l'engin de données numéro 1. L'application Android ne sera pas au courant de cette forme de redirection de la requête vers cet engin numéro 1. Traefik dans le Docker Compose se chargera de régler cette redirection.

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

```
{
   "chaine": string   // chaîne de caractères (en UTF-8) pour la recherche
}
```

L'engin de données retournera la liste des stations dont les noms satisfont le patron de la chaîne de caractères. La liste peut être vide ou contenir un ou plusieurs éléments.

```
{
   "stations" : [
      {
         "code": integer,   // code unique associé à la station
         "nom":  string     // nom de la station
      },
      . . .
   ]
}
```

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)

`GET /station/<code>`

Retourne toute l'information sur la station ayant l'identifiant <code> pour être capable d'obtenir les coordonnées permettant de préciser graphiquement sa position sur une carte. On supposera qu'une carte existe (dans l'application Android ou ailleurs) et pourra être utilisée pour la visualisation. Cette requête sera également prise en charge par l'engin numéro 1.

```
{
     "nom":  string,
     "latitude":   float,
     "longitude":  float
}
```

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

Statuts HTTP: 200 : ok
400 : mauvaise requête
404 : pas de station pour le code spécifié

`GET /donnees/usage/<temps>/<station>`

Cette requête peut être utilisée pour accéder à certaines données, brutes et sous forme de graphiques pour les présenter à l'usager de façon appropriée dans l'application Android. <temps> peut prendre les valeurs : "parheure", "parjourdelasemaine" ou "parmois" pour indiquer le regroupement par unité de temps d'observation choisi (axe x). <station> peut représenter un code de station valide ou prendre la valeur "toutes" lorsque l'ensemble des stations doit être pris en compte ou seulement une en particulier.

Cette requête sera prise en charge par l'engin de données numéro 2 puisque cet engin se concentre sur le retour de statistiques de base et de graphiques à fournir à l'application Android même si celle-ci ignore toujours quel engin lui fournit ces données précisément.

Le JSON retourné par l'engin de données aura la structure suivante:

```
{
   "donnees" : [
      [
         integer,  // selon la caractéristique de temps (axe x)
         integer   // nombre de départs de parcours Bixi (axe y)
      ],
      . . .
   ]
   "graphique": string   // PNG ou JPEG du graphique avec axes bien identifié
                         // dans l'image même – attention, peut être gros!
}
```

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

Statuts HTTP: 200 : ok
400 : mauvaise requête
404 : pas de données pour la requête effectuée

`GET /prediction/usage/<station>`

Cette requête est utilisée pour lancer un algorithme d'apprentissage par forêt aléatoire visant à prédire l'usage d'une station donnée à partir des jours de la semaine, de l'heure, du jour et de la température. <station> peut représenter un code de station valide ou prendre la valeur "toutes" lorsque l'ensemble des stations doit être pris en compte ou seulement une en particulier. On peut considérer les données de 2017 comme étant celles qui ne participent pas à l'apprentissage et qui sont plutôt considérées pour vérifier les résultats obtenus.

Cette requête sera prise en charge par un engin de données séparé, le numéro 3, qui se concentre uniquement sur les prédictions.

Le JSON retourné par l'engin de données aura la structure suivante:

```
{
   "donnees" : [
      [
         integer,  // selon la caractéristique de temps (axe x)
         integer   // nombre de départs de parcours Bixi (axe y)
      ],
      . . .
   ]
   "graphique": string   // PNG ou JPEG du graphique avec axes bien identifié
                         // dans l'image même – attention, peut être gros!
}
```

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

Statuts HTTP:	200 : ok
	400 : mauvaise requête
	404 : pas de données pour la requête effectuée

`GET /prediction/erreur/`

La requête est utilisée pour retourner un graphique et un tableau de valeur montrant la différence entre les valeurs prédites et celles de 2017 pour les mêmes dates. Ceci permettra de juger de la qualité du modèle d'intelligence artificielle obtenu lors de l'apprentissage.

Cette requête est aussi prise en charge par l'engin de données numéro 3.

Si le système n'a pas terminé sa phase d'apprentissage ou n'a même pas débuté, la requête retournera un code d'erreur approprié. Si le système peut évaluer l'erreur, il retournera la structure générale JSON suivante :

```
{
   "donnees" : [
      [
         integer,  // temps (axe x)
         float     // différence de valeurs prédites versus réelles (axe y)
```

```
      ],
      . . .
   ]
   "graphique": string   // PNG ou JPEG du graphique avec axes bien identifié
                         // dans l'image même – attention, peut être gros!
}
```

Statuts HTTP: 200 : ok
400 : mauvaise requête (erreur dans le JSON envoyé)
404 : aucun apprentissage effectué ou encore disponible

`GET /logs/<engin>/<dernierOctet>`

Pour que l'application PC puisse obtenir une partie des messages d'un engin de données. <engin> peut prendre la valeur 1, 2 ou 3 pour identifier l'engin de données en question duquel on désire obtenir des messages. Il appartiendra à Traefik d'aiguiller la requête au bon engin.

<dernierOctet> est un nombre entier représentant le nombre d'octets des messages que l'application PC a reçus cumulativement jusqu'au moment où la réponse à une requête de ce type a été reçue pour la dernière fois. Ceci implique qu'on évoquera cette requête pour la première fois avec ce paramètre à zéro (« 0 ») et qu'il augmentera à chaque nouvelle requête par la suite. L'engin devra donc retourner uniquement le contenu des messages accumulé depuis <dernierOctet> dans la réponse à la requête. À noter que mes messages peuvent inclure des données sous forme textuelle mais aussi des graphiques.

Le mécanisme d'authentification de base doit être utilisé avec cette requête.

Le serveur central retournera un vecteur de données dans un JSON s'il y en a à retourner. On indiquera la quantité :

```
{
   "nbrOctets": integer,   // nombre total d'octets dans cette réponse
   {
      "texte" : true|false, // le contenu est sous forme de texte (true) ou
                            // de graphique (en PNG probablement) (false)
      [ "ligne ou graphique", "ligne ou graphique", ... ]
   },
   . . .
]
```

Le serveur ne retournera aucun JSON s'il n'y a pas de nouveaux messages à retourner, mais le signifiera par un code HTTP approprié (204) à l'application PC

Statuts HTTP: 200 : ok
204 : aucun contenu à retourner
401 : utilisateur non authentifié

Contenu des livrables

BIXI demande deux phases de développement en plus de la démonstration d'un prototype au préalable. Chaque étape devra offrir un système fonctionnel, au moins partiellement, selon les critères explicités dans cette section des exigences techniques.

Prototype

Pour un fonctionnement minimum de base du **prototype** le jeudi 24 septembre:

- Communication de base entre l'application Android et le serveur web avec interface REST; (un message de log ou autre, par exemple)
- Serveur dans un conteneur, mais sans système Docker Compose.
- Code Python de l'application de prédiction des températures de l'article reproduit avec maîtrise de la compréhension des opérations. Pas de communications avec le reste du système pour l'instant.

Livrable intermédiaire

Pour le **livrable intermédiaire**, évalué le jeudi 29 octobre, BIXI s'attend aux fonctionnalités suivantes :

- Serveur web:
  - Tout ce qui touche le sondage;
- Engins de données :
  - Consultation des statistiques;
  - Toutes les données et analyses complétées;
  - Retourner tous les résultats à l'application Android;
  - Des logs produits localement en sortie standard par les engins;
- Application Android :
  - Toutes les différentes vues;
  - Il peut manquer les résultats de prédiction (engin de données 3) à afficher;
- Application sur PC :
  - Consultation des résultats du sondage;
  - Préparation de l'interface usager pour recevoir les logs des engins de données, mais sans la réception des logs eux-mêmes;

Livrable final

Au **livrable final**, évalué le mardi 1er décembre, vous devez terminer le projet, ce qui implique, par rapport au livrable intermédiaire :

- Serveur web:
  - Vérification de la perte d'un engin de données;
  - Serveur dans un conteneur Docker avec Traefik et Docker Compose;
- Engin de données :

    - Recherche et visualisation d'une station Bixi sur la carte;
    - Raffinement de la prédiction à un niveau raisonnable;
    - Perte des engins de données fonctionnelle;
    - Engins dans des conteneurs et démarrés avec Docker Compose;
- Application Android :
    - Compléter les vues manquantes;
    - Tenir compte des remarques de l'évaluation du livrable intermédiaire pour ce qui concerne l'esthétisme et l'ergonomie de l'interface et effectuer les modifications demandées;
- Application PC:
    - Réception des logs des engins de données complétée;

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