25/12/2017

Leeroy

Département du génie informatique

AccessTALK’s

DOCUMENT DE D’ANALYSE ET DE CONCEPTION

Table des matières

[**I-** **DESCRIPTION DU PROJET** 2](#_Toc502051016)

[A- Présentation générale 2](#_Toc502051017)

[B- Exigences fonctionnelles 2](#_Toc502051018)

[C- Exigences non fonctionnelles 4](#_Toc502051019)

[**II-** **ANALYSE DU PROBLÈME** 4](#_Toc502051020)

[A- Structure du fichier log 4](#_Toc502051021)

[B- Analyse du problème 6](#_Toc502051022)

[**III-** **ETUDE TECHNIQUE** 7](#_Toc502051023)

[A- Choix de la technologie 7](#_Toc502051024)

[B- Structures de données 8](#_Toc502051025)

[C- Séquencement des activités 9](#_Toc502051026)

[1- Attente d’une nouvelle ligne 9](#_Toc502051027)

[2- Récupération des informations d’une ligne 9](#_Toc502051028)

[3- Envoie des informations 11](#_Toc502051029)

[4- Sauvegarde des informations 11](#_Toc502051030)

[**IV**- **DESCRIPTION DU CODE SOURCE** 16](#_Toc502051031)

# **DESCRIPTION DU PROJET**

## Présentation générale

Il s’agit de concevoir un outil de présentation temps réel du contenu du fichier log de l’outils de mise en cache “Squid”, via une plateforme web.

## Exigences fonctionnelles

La plateforme doit exploiter les informations contenues dans ce fichier log, afin de les présenter de manière condensée sous forme de graphes :

* Les informations temps réel :
  + Les différents clients connectés, pondérés par leurs nombres de requêtes émises
  + Les différents urls sollicités, pondérés par le nombres de requêtes les concernant
  + Les différents types de ressources sollicités, pondérés par le nombre de requêtes les concernant
  + Le nombre total de requêtes émises
  + Le nombre total de requêtes émises, concernant une ressource effectivement mise en cache par l’outils de mise en cache (requêtes dite HIT)
  + Le nombre total de requêtes émises, concernant une ressource non disponible dans la mémoire cache de l’outils de mise en cache (requêtes dite MISS)
  + La taille totale des données téléchargés
  + Le temps moyen de traitement d’une requête par le serveur
* Les informations relatives à une journée antérieure :
  + Les différents clients connectés, pondérés par leurs nombres de requêtes émises
  + Les différents urls sollicités, pondérés par le nombres de requêtes les concernant
  + Les différents types de ressources sollicités, pondérés par le nombre de requêtes les concernant
  + Le nombre total de requêtes émises
  + Le nombre total de requêtes émises, concernant une ressource effectivement mise en cache par l’outils de mise en cache (requêtes dite HIT)
  + Le nombre total de requêtes émises, concernant une ressource non disponible dans la mémoire cache de l’outils de mise en cache (requêtes dite MISS)
  + La taille totale des données téléchargés
  + Le temps moyen de traitement d’une requête par le serveur

Outre la consultation de ces informations, l’utilisateur de l’application devra être en mesure de :

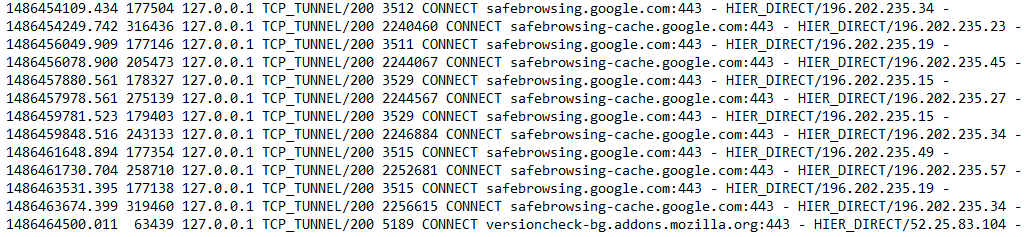
* Configurer le chemin d’accès au fichier log, source d’informations à présenter.
* Modifier la période de rafraichissement des informations temps réel (appelé dans ce document période du temps réel)

## Exigences non fonctionnelles

* Lecture temps réel du fichier
* Traitement rapide des données
* Déploiement sur un environnement Linux

# **ANALYSE DU PROBLÈME**

## Structure du fichier log

Il s’agit d’un problème de collecte et de présentation d’informations contenu dans un fichier log. La structure du dit fichier est la suivante :

Colonnes après colonnes, on a :

1. Le moment : il s’agit de la date à l’instant où la requête est envoyée sous le format UNIX (Unix timestamp). Cette date est en milliseconde et possède trois chiffres après la virgule.
2. Durée : le temps de traitement de la requête par le serveur. Il diffère dans l'interprétation entre HTTP et ICP :

* Pour HTTP, il s'agit essentiellement du temps écoulé entre la réception de la requête et le moment où Squid a fini d'envoyer le dernier octet de la réponse.
* Pour ICP, c'est le délai entre la planification d'une réponse et son envoi.

3. Adresse du client : il s’agit de l’adresse IP de l’instance de la requête.

4. Code du résultat : constitué de deux entrées séparées par un slash.

5. Bytes : il s’agit de la taille des données délivrées au client. Cela ne constitue pas la taille nette de l'objet, car les en-têtes sont également comptés.

6. Type de requête : il s’agit du type de la requête qui a été envoyée pour obtenir la ressource.

7. URL : il s’agit de l’URL demandée. À noter que le fichier journal peut contenir des espaces pour l'URL

8. L’utilisateur : la huitième colonne peut contenir l'identité de l'utilisateur pour le client demandeur. Si aucune identité d'utilisateur n'est disponible, un "-" sera enregistré.

9. Un code permettant de savoir comment la requête a été traitée. Ce code peut être suivi par l'adresse IP vers laquelle la requête a été redirigée.

10. Le type de contenu issu du header HTTP de la réponse. Il s’agit donc du type de la ressource sollicité.

## Analyse du problème

Tandis que squid fonctionne, le fichier log reçoit épisodiquement (en fait au moment précis où les requêtes sont envoyées et les réponses délivrées) de nouvelles lignes, relative chaque requête. Pour chaque nouvelle ligne, nous récupèrerons les différents éléments présentés plus haut. Nous devons présenter les informations en temps ; il est donc nécessaire de traiter les nouvelles lignes le plus tôt possible. Nous les traiterons dès l’instant de leur arriver sur le fichier.

Après avoir récupérer les différentes informations d’une ligne, il convient de les envoyer à la vue d’une part, puis de les stocker de sorte que l’on puisse les consulter ultérieurement (par jour/semaines). Cependant, le fait que l’utilisateur peut modifier la valeur du temps réel nous empêche d’envoyer directement les informations d’une nouvelle ligne au navigateur ; cela suggère en effet que nous récoltions ces informations là pour toutes les nouvelles lignes, jusqu’à ce que le laps de temps représentant le temps réel soit épuisé, moment auquel on enverra l’ensemble des informations récoltés.

Par ailleurs se pose le problème de la nature de la structure de donnée qui contiendra ces informations, prêts à être affiché, dans le programme. Quoiqu’il en soit, on notera deux principales unités de stockage :

* L’unité dite volatile contiendra les informations recueillies pendant le laps de temps représentant le temps réel, qui sera réinitialisé après chaque laps de temps qui représente le temps réel c’est-à-dire après chaque envoie à la vue. Cette unité sera uniquement présente dans l’application, et ne sera pas persistante.
* L’unité dite persistante, qui contiendra toutes les informations jamais recueilli par l’application, ce qui permettra de présenter les informations des jours / semaines antérieurs. Cette unité sera en partie dans l’application, et sera sauvegardé en dehors de l’application.

# **ETUDE TECHNIQUE**

## Choix de la technologie

Il existe une technologie web idéale pour les communications asynchrones entre serveur et client, sans recharge de la page : socket.io. Nous l’utiliserons pour l’envoie des informations en temps réel au navigateur. Par ailleurs les langages permettant une implémentation aisée de socket.io sont *php, python, java ee et NodeJS*.

Afin d’éviter des files d’attente de lignes à traité qui nous feraient perdre du temps et empièteraient sur notre temps réel, il convient mieux d’utiliser un langage évènementiel, qui traitera sur des threads différents chaque nouvelles ligne (l’arrivée d’une nouvelle ligne étant un évènement…).

*NodeJS* est donc le choix le plus judicieux.

## Structures de données

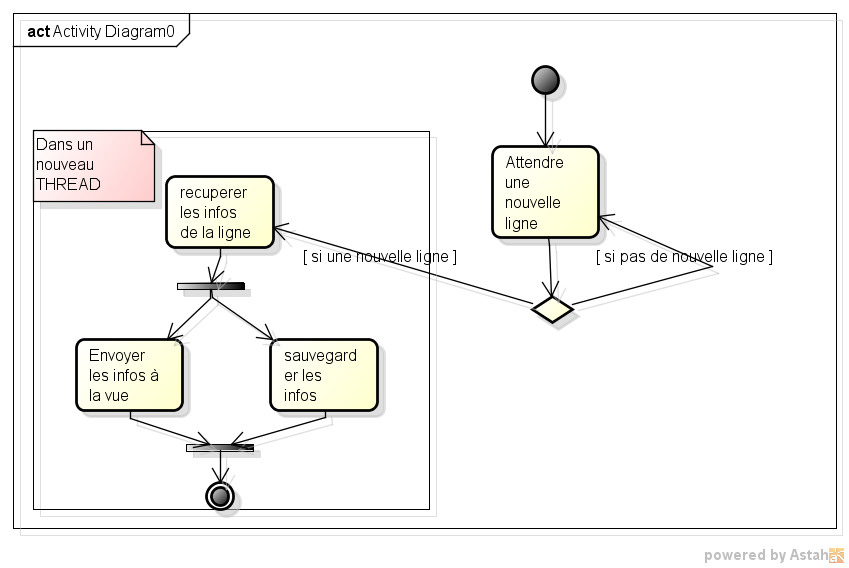
Comme mentionné plus haut, nous devrons trouver la structure de donnée adéquate pour contenir les informations récupérées de la nouvelle ligne, et à envoyer au navigateur pour affichage ; Il doit donc s’agir d’une structure de donnée simple, pas trop lourde, qui plus est facile et rapide à manipuler avec notre langage de programmation NodeJS, et avec le javascript qui se chargera de l’utiliser pour remplir les graphes du coté vue. Aussi, les opérations de persistance (de la variable), et de récupération des données persistées afin de retrouver la variable, ne doivent pas être lourd. Le JSON réponds parfaitement à ces 04 conditions.

En effet, le JSON est la structure de donnée par excellence du Javascript, langage pour lequel NodeJS est une extension. Donc JSON est ultra simple et rapide à manipuler avec Javascript / NodeJS.

Aussi, nous utiliserons les fichiers pour faire persister ces données ; Ce choix ne pose aucun problème à JSON qui à la base est une structure de fichier (et a été étendu en variable javascript pour faciliter la manipulation des données en javascript).

Ce choix est justifié par le fait qu’une base de données est trop lourde pour sauvegarder plusieurs instances d’un même et unique type d’information.

## Séquencement des activités

Maintenant que nous avons fixé les outils à utiliser, il ne reste plus qu’à définir le séquencement des activités du programme. Globalement, le fonctionnement du programme sera le suivant :

### Attente d’une nouvelle ligne

À ce niveau, il s’agit principalement d’attacher à l’évènement « nouvelle ligne » sur le fichier, la fonction de récupération des informations.

### Récupération des informations d’une ligne

Chaque nouvelle ligne est traité dans un Thread nouveau. La variable qui récupère les informations de ladite ligne est dataLive, une variable json dont la structure est la suivante :



Ainsi, pour toutes les lignes traitées pendant le laps de temps représentant le temps réel (une ligne pour une requête), on a :

* SommeTempsTraitement contient la somme du temps de traitement de chacune des requêtes, en milliseconde.
* nombreRequettesHit / nombreRequettesMiss le nombre de ligne correspondant aux requetes Hit / Miss.
* SommeNombreMega est la somme de la taille des données réponse du serveur pour chacune de ces requêtes.
* Client contient la liste des clients impliqués dans ces requêtes, avec leur nombre d’occurrence.
* Ressource contient la liste des types de ressources impliqués dans ces requêtes, avec leur nombre d’occurrence.
* Site contient la liste des urls impliqués dans ces requêtes, avec leur nombre d’occurrence.

Une fois le laps de temps représentant le temps réel écoulé, les informations contenues dans dataLive sont sauvegardées, envoyé au navigateur, puis la variable est réinitialisée et le cycle recommence.

### Envoie des informations

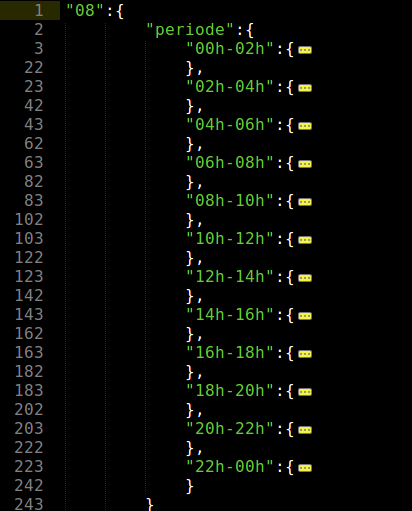
Afin d’envoyer les informations en temps réel, nous programmons juste l’exécution de la fonction d’envoie après chaque intervalle de temps de durée égal à la période du temps réel. Ainsi, après chaque seconde (par exemple), la variable dataLive est envoyée au client, puis réinitialisée, le tout de manière automatique.

### Sauvegarde des informations

Afin de faciliter l’opération de restauration de l’information à partir des données sauvegardées, nous choisissons de représenter les données stockées tels qu’elles seront utilisées pour l’affichage, et telle qu’elles ont été récupérées. Aussi nous choisissons de séparer les fichiers de sauvegarde par semaines ; ceci afin de limiter le nombre de fichiers dans le serveur, et de limiter la taille d’un fichier. De cette façon, l’application n’ajoutera les données du dataLive que dans le fichier de la semaine contenant le jour en cour. Un fichier de sauvegarde sera juste un fichier json (au format JSON) dont le contenu sera similaire à ce qui suit :



* Le champs nomFichier aura la même valeur que le nom du fichier, et représente la date du lundi de la semaine que représente le fichier.
* Tous les nombres qui suivent sont des entrés correspondant au jour dont le numéro dans le mois est le nombre. Ils contiennent les données du jour en question. Leur contenu est semblable à ceci :



* Le jour est scindé en période. Chaque période contient l’ensemble des informations des dataLive de la période. Le contenu d’une période est le suivant :



* Toutes les informations présentes ici sont la fusion de toutes les informations rapportées par les dataLive de la période. Ainsi, le client 192.168.10.191 a envoyé au total 14 requêtes entre 16h et 18h le 08/05/2017.

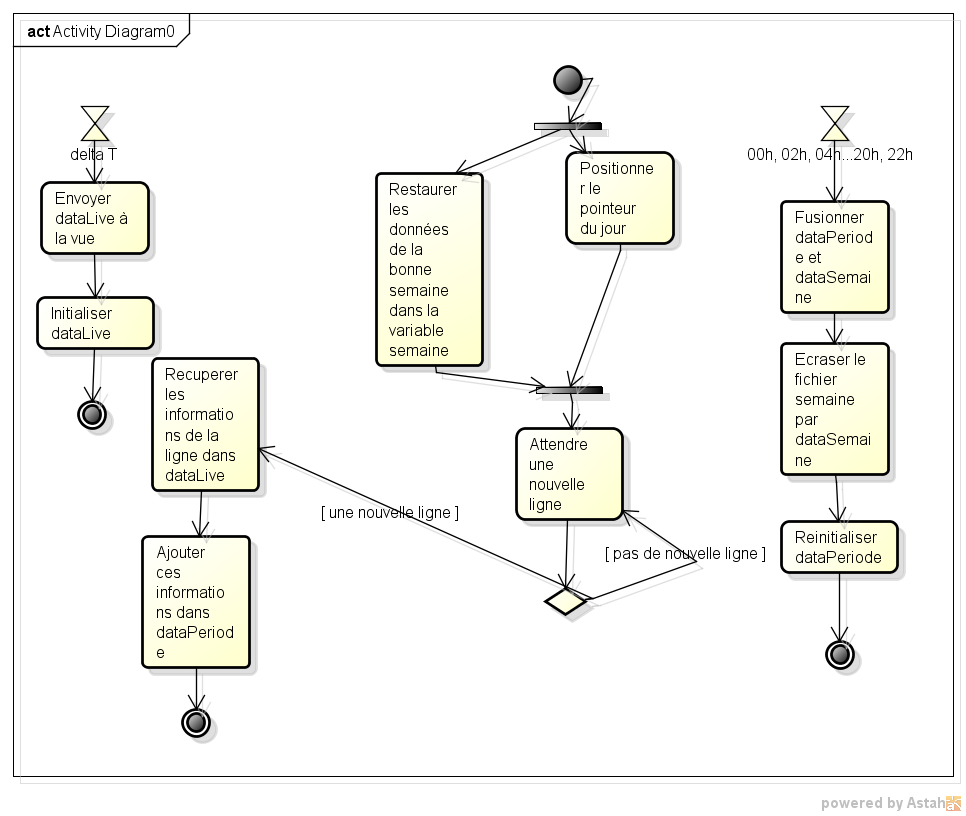
En résumer, pour sauvegarder un datalive, l’application devra :

* Retrouver le bon fichier de sauvegarde
* Retrouver le bon jour dans le fichier de sauvegarde
* Retrouver la bonne période dans le bon jour
* Ajouter les données du dataLive aux données de la période
* Écraser le fichier avec le nouvel ensemble.

Afin d’éviter ce long traitement après chaque laps de temps réel, nous optons pour une sauvegarde périodique des données. Ainsi nous enregistrerons les données à des heures fixe de la journée : 00h, 02h, 04h…. 20h et 22h. À noter que la sauvegarde se fait de période en période et pas d’instant en instant (instant de temps réel) parce qu’il serait très compliqué d’afficher dans un graphe des informations pour tout instants d’une journée ; il est par contre plus simple de le faire pour toutes les deux heures, ainsi on a que 12 points sur l’axe des abscisses.

Ainsi, l’on entassera préalablement toute ces données dans une variable appelé dataPeriode, qui sera enregistré puis réinitialisé toutes les deux heures.

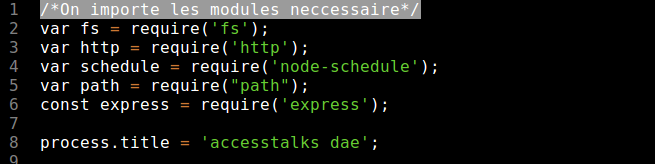
Au démarrage, l’application récupèrera donc la sauvegarde de la semaine correspondante, et initialisera un pointeur « jour » vers l’entrée correspondante au jour actuel dans la variable semaine. Ainsi, aux heures de sauvegarde, il suffira de remplacer le contenu de la période correspondante du jour pointé dans la variable semaine par le contenu de la variable dataSemaine, et écraser le fichier de la semaine chargé plus haut. On aura donc le séquencement suivant :



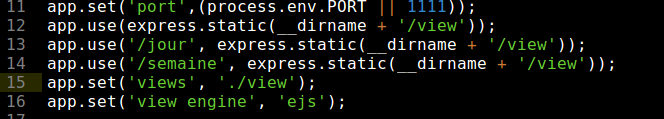
À noter que lors de la récupération des données de la semaine en cours, si le fichier semaine n’existe pas, il est alors créé et vide.

# DESCRIPTION DU CODE SOURCE

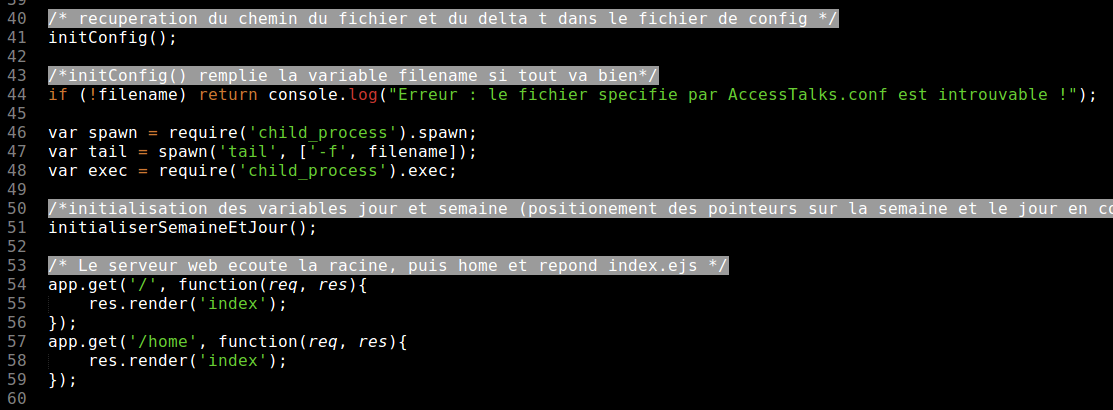
* On importe les modules nécessaires



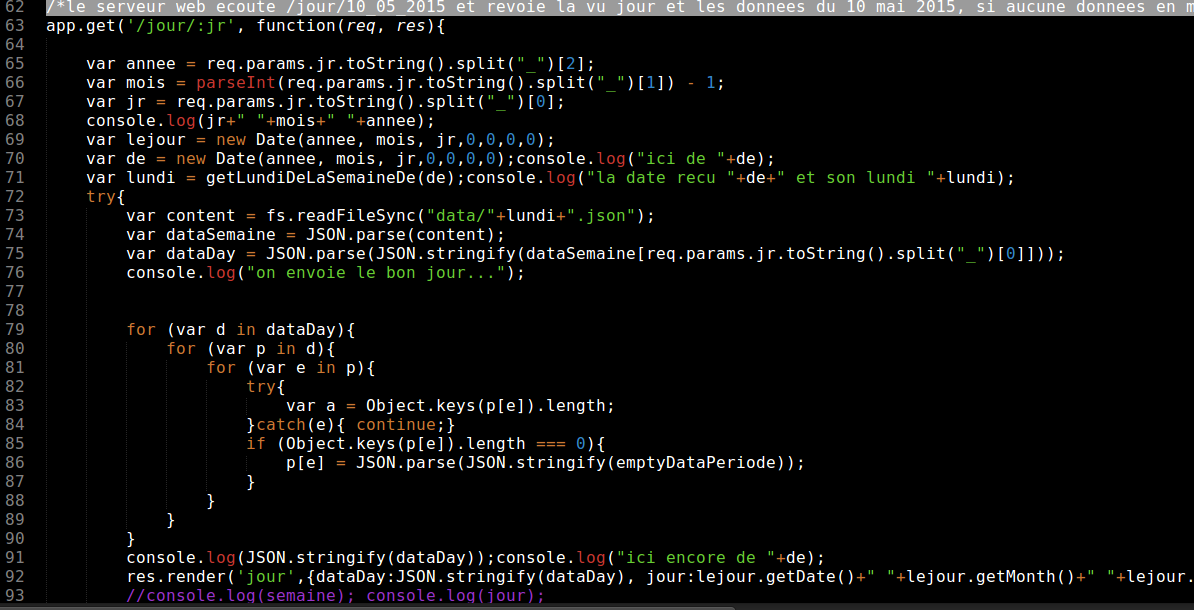
* On défini le port d’écoute et les répertoires où l’application ira chercher les vues



* On récupère les configuration (valeur du temps réel, chemin vers le fichier log) actuel, on restaure les variables jour et dataSemaine, et on défini la vue à renvoyer à la racine de l’application web.



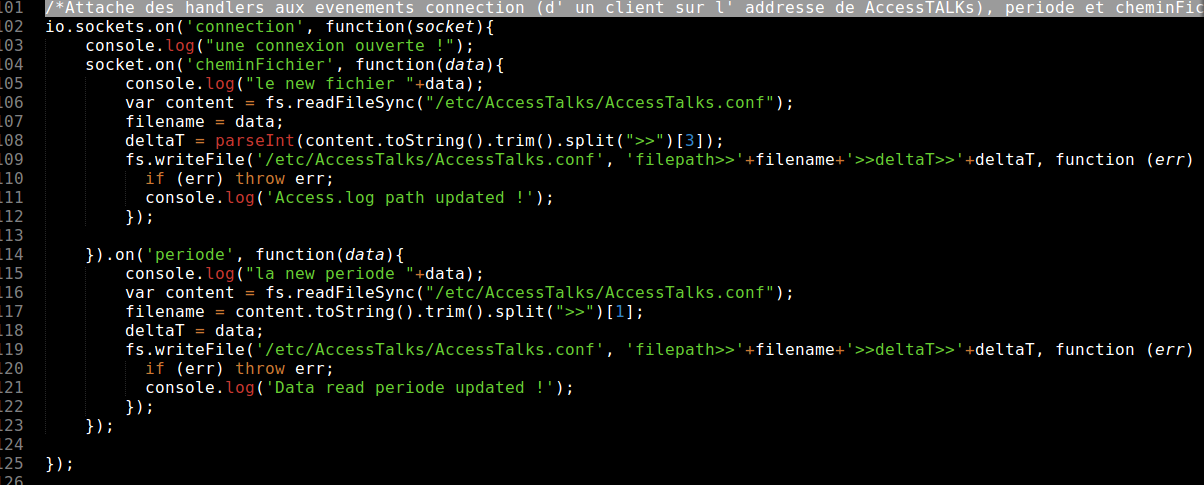
* On traite la demande de consultation d’informations par jour



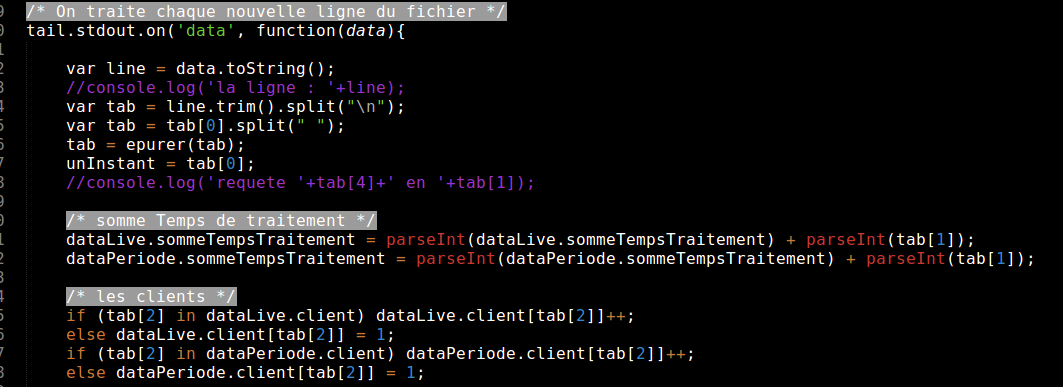
* On crée le serveur web, puis on initialise les variables dataLive et dataPeriode.



* On attache des réactions aux tentatives de modification des configurations de l’application

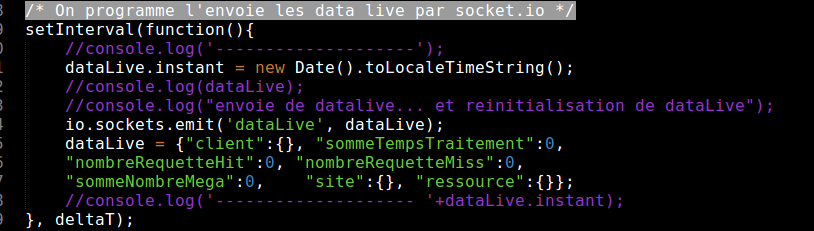


* On attache à l’évènement « nouvelle ligne » le traitement correspondant





* On programme l’envoie periodique du dataLive



* On programme la sauvegarde du dataPeriode à des heures fixe



Voilà ce qui a été fait jusqu’à présent.