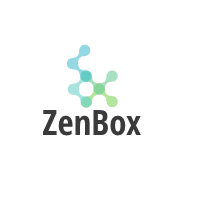


27/04/2018

Aguirre Max & Izabelle François & Guilpain Léo

Délivrable 3



*« Nous attestons que ce travail est original, qu’il indique de façon appropriée tous les emprunts, et qu’il fait référence de façon appropriée à chaque source utilisée »*

Table des matières

[Introduction 2](#_Toc515026622)

[Rappel du projet 3](#_Toc515026623)

[A- Gantt 3](#_Toc515026624)

[B- Les Étapes de notre projet 3](#_Toc515026625)

[Rapport d’avancement 5](#_Toc515026626)

[A- Explication des réalisations 5](#_Toc515026627)

[1. Création de la base de données (voir annexe 1) 5](#_Toc515026628)

[2. Insertion et connexion à la base de données (voir annexe 2) 6](#_Toc515026629)

[3. Sélection des données dans la base (voir annexe 3) 7](#_Toc515026630)

[4. Serveur (voir annexe 4) 8](#_Toc515026631)

[5. Relation serveur et traitement du langage (voir annexe 5) 9](#_Toc515026632)

[6. Gestion des commandes de l’utilisateur (voir annexe 6) 10](#_Toc515026633)

[7. Thread (voir annexe 7) 10](#_Toc515026634)

[8. Résultat des commandes sur Eclipse 13](#_Toc515026635)

[9. Résultat des commandes sur le terminal 13](#_Toc515026636)

[10. Partie React 13](#_Toc515026637)

[B- Difficultés rencontrés 15](#_Toc515026638)

[Pour la suite 16](#_Toc515026639)

[A- Travaux possible 16](#_Toc515026640)

[B- Solution finale espérée 17](#_Toc515026641)

[Apport du projet 17](#_Toc515026642)

[Conclusion 18](#_Toc515026643)

[Annexes 19](#_Toc515026644)

[Annexe 1 : Création de la base de données 19](#_Toc515026645)

[Annexe 2 : Insertion et connexion à la base de données 20](#_Toc515026646)

[Annexe 3 : Sélection des données dans la base 21](#_Toc515026647)

[Annexe 4 : Serveur 22](#_Toc515026648)

[Annexe 5 : Relation serveur et traitement du langage 23](#_Toc515026649)

[Annexe 6 : Gestion des commandes de l’utilisateur 24](#_Toc515026650)

[Annexe 7 : Thread 25](#_Toc515026651)

Introduction

Au cours de notre deuxième année en ESIR 2 nous avons été amenés à travailler sur un projet par groupe de 3. Notre objectif est de concevoir une box domotique avec laquelle on peut interagir via une chatbox textuelle, le tout sans héberger de données dans le cloud. Depuis le début de l’année nous avons effectué des recherches afin de prendre connaissance des projets de ce type déjà réalisés. Nous avons également fourni un état de l’art concernant tous les produits de la sorte qui ont été commercialisés. Le but était de fournir un plan de notre travail afin de savoir de manière plus précise ce que nous allions faire. Dans cette optique de planification nous avons fourni plusieurs délivrables afin de rendre compte de notre avancement et de ce que nous comptions réaliser, et comment nous comptions le réaliser. Dans ce troisième délivrable, nous allons vous rappeler les objectifs du projet de manière plus précise, notre Gantt et quelles ont été les étapes que nous avons suivies afin d’arriver là où nous en sommes. On fera ensuite un rapport d’avancement en expliquant tout ce que nous avons réalisé de manière plus précise en expliquant notre architecture et notre code, en détaillant les difficultés rencontrées.

Rappel du projet

Une image contenant capture d’écran

Description générée avec un niveau de confiance très élevéA- Gantt

Figure 1 : Diagramme de Gantt de notre projet

B- Les Étapes de notre projet

Une fois les premiers délivrables de faits, nous avions un plan précis concernant notre projet. Nous avons commencé par nous renseigner spécifiquement sur Core NLP afin de pouvoir définir les différentes fonctionnalités qui allaient nous intéresser et nous avons tenté plusieurs approches afin de savoir les utiliser. Premièrement nous avons voulu utiliser CoreNLP via un wrapper node js nous permettant de l’utiliser directement dans node js. Nous avions une version très basique de notre système de traitement du langage et nous nous sommes aperçus qu’une des fonctionnalités importantes de CoreNLP, qui aurait pu nous permettre de perfectionner notre projet et de nous amener là où nous voulions aller, n’était pas implémentée par ce wrapper. À la suite de ça nous avons envisagé d’utiliser un autre système de traitement du langage, plus simple à prendre en main. Nous nous sommes renseignés et nous nous sommes rendu compte un peu tard que cette nouvelle solution ne pouvait être utilisée en local et que ce système devait être hébergé sur un cloud.

Après cette période nous avons décidé de nous pencher à nouveau sur la solution de base que nous avions choisie : CoreNLP. Nous avons décidé de l’utiliser directement en Java puisque c’est sous ce langage de programmation que ce système a été développé, et donc toutes les fonctionnalités y étaient implémentées et utilisables. À partir de là nous sommes arrivés à une solution basique, où il nous était possible d’analyser une phrase et d’en ressortir les composants. Afin d’entraîner ce système nous avons créé une base de données dans laquelle nous avons ajouté les verbes qu’un utilisateur pour. En même temps, nous avons développé le côté serveur de notre solution et nous sommes arrivés à une première version à partir de laquelle nous étions capable de récupérer les commandes utilisateurs et les transmettre aux objets connectés.

En parallèle de ça nous avons travaillé à allumer une lampe ikea en passant par la gateway. Pour cela nous avons installé une image docker sur laquelle tout était préconfiguré pour parler avec la gateway ikea en protocole Coap. Après avoir réussi à allumer la lampe il nous fallait pouvoir relier le docker à notre système. Après avoir fait cela nous avons obtenu une version dans laquelle l’utilisateur pouvait envoyer une commande au système, puis la lampe s’allumait et s’éteignait. Cependant la commande était très longue avant d’être prise en compte. C’est à dire que lorsque l’on envoyait une commande via l’interface textuelle, il fallait que notre serveur de traitement du langage se lance (ce qui prenait une dizaine de secondes), ainsi que notre docker. Au total, il fallait attendre une quinzaine de secondes avant que la commande soit prise en compte.

Afin de palier à ce problème nous avons décidé de travailler à faire tourner le système de traitement de langage en continu via un second process, ce qui nous éviterait de le relancer à chaque fois. De plus nous avons appris que le logiciel de traitement de langage pouvait utiliser plusieurs algorithmes différents afin de procéder à l’analyse d’un texte. Certains sont plus rapides que d’autres mais moins précis, ce qui n’est pas un problème pour notre système puisque son fonctionnement ne repose pas sur une analyse trop profonde de la syntaxe. Nous avons donc modifié l’algorithme utilisé afin que notre système fonctionne plus rapidement. Grâce à ces modifications, nous avons réussi à gagner environ 10 secondes sur l’allumage de la lampe.

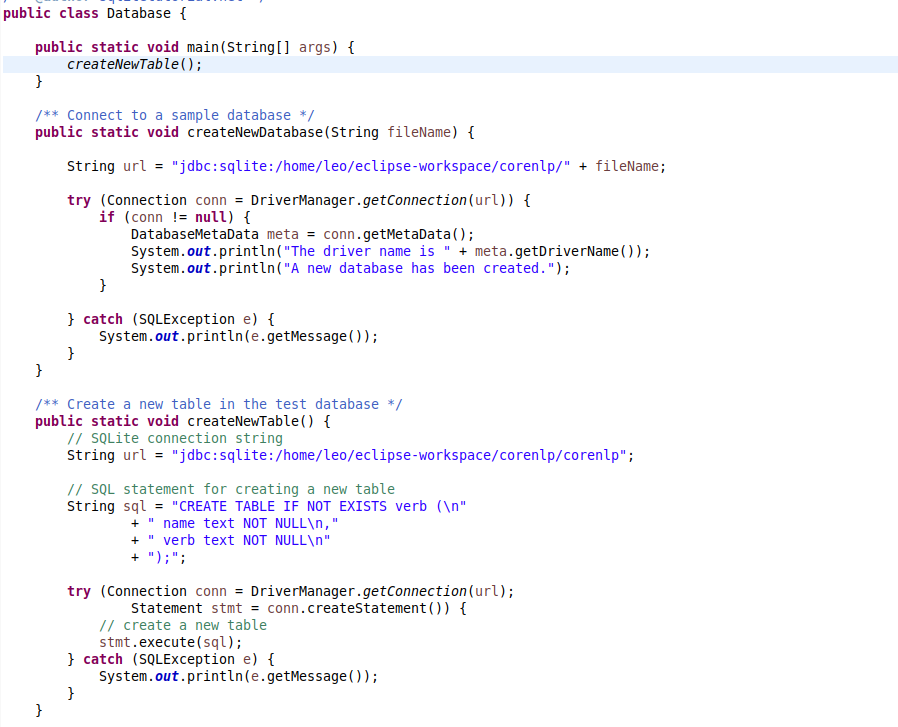
Rapport d’avancement

A- Explication des réalisations

Les codes seront disponibles dans leur intégralité en annexe mais également sur le Github.

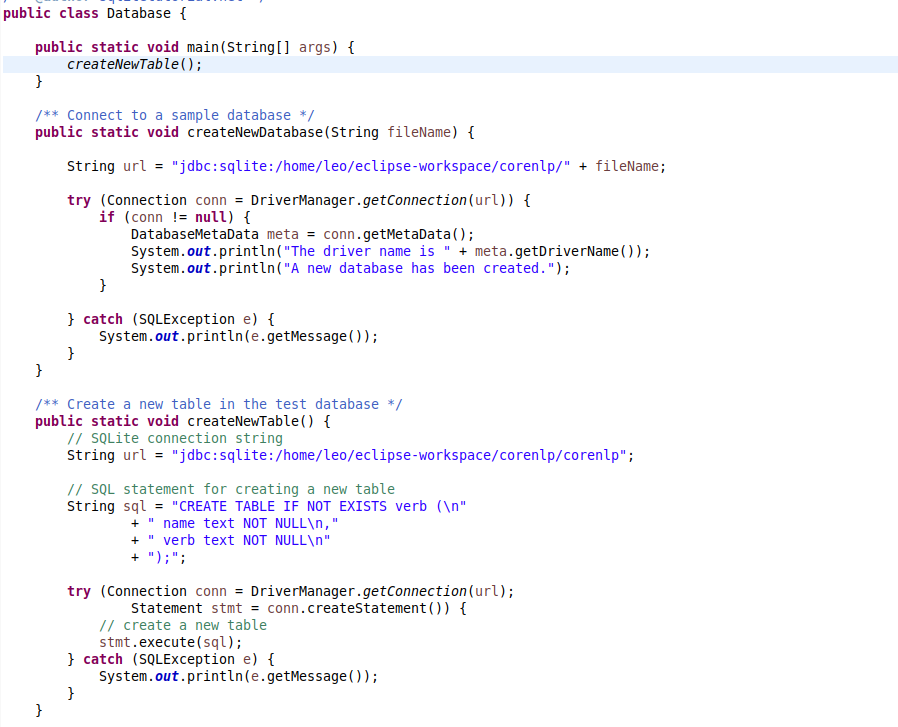
1. Création de la base de données (voir annexe 1)

Nous allons créer notre base de données. Cette base servira à entraîner notre système.



Dans ce code, nous avons créé la base de données que l’on a nommé “corenlp”.

Ensuite, nous avons créé une table :

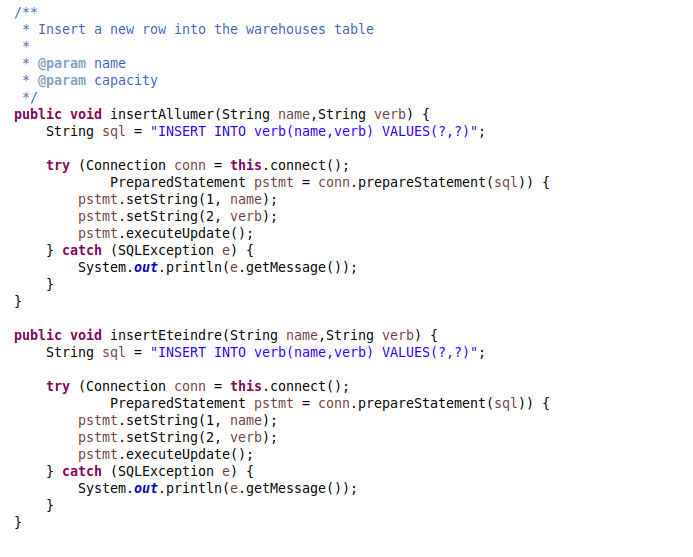


La table que nous avons nommée ”verb” contient deux colonnes. Une contenant le nom de la commande à effectuer et l’autre contenant le verbe. Cela permet d’associer une commande à un verbe.

2. Insertion et connexion à la base de données (voir annexe 2)



Ce code permet de se connecter à la base de données “corenlp”.

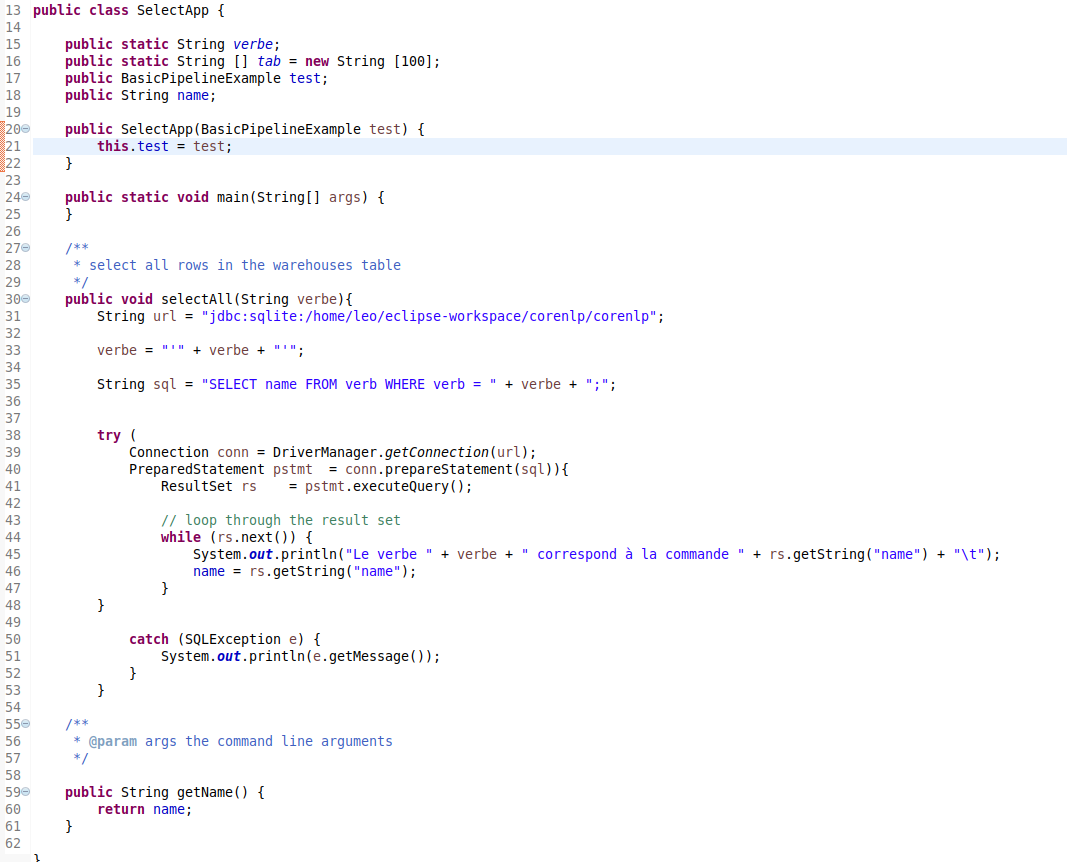


Ce code permet l’insertion dans la base de données. D’après la requête SQL, on insère dans la table ”verb” deux valeurs.



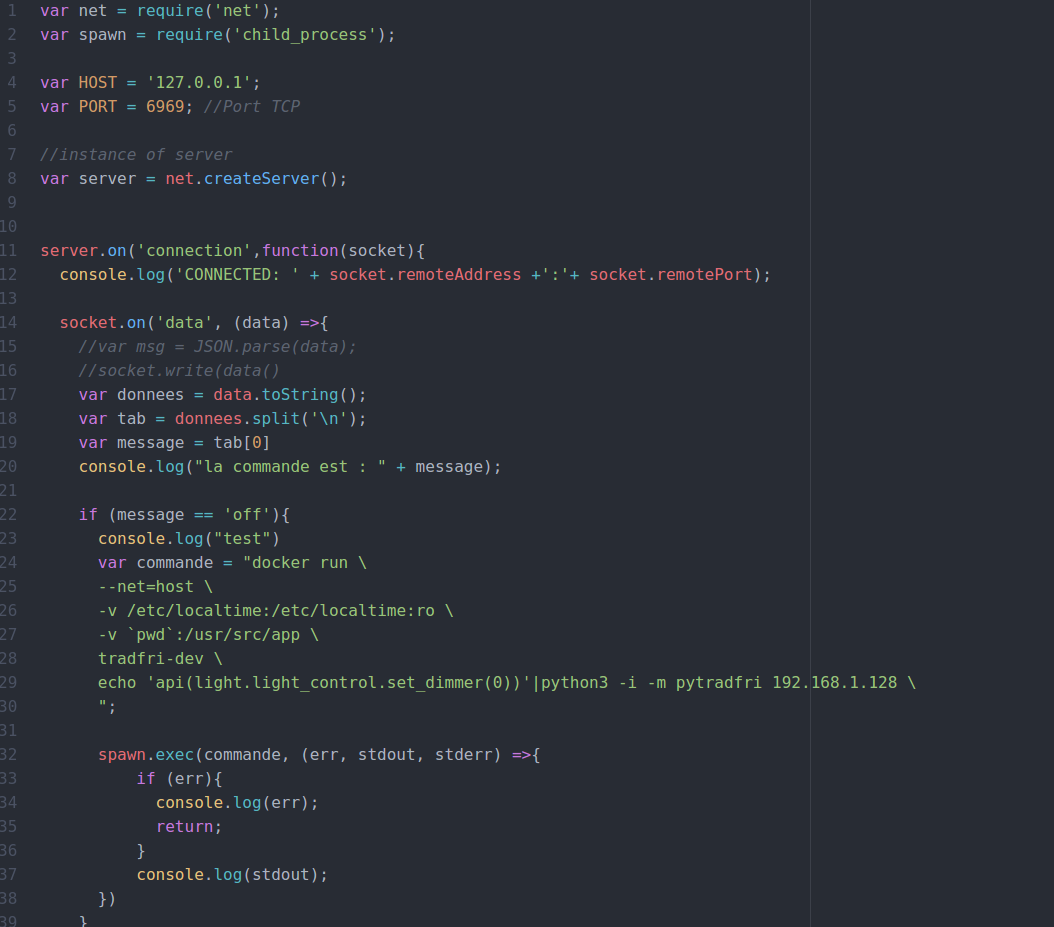
Après avoir réalisé les différentes fonctions il faut maintenant ajouter les valeurs dans la base de données. Ces valeurs représentent l'entraînement de notre système. Par exemple “activate” correspond à la commande “on” tandis que “down” correspond à la commande “off”.

3. Sélection des données dans la base (voir annexe 3)

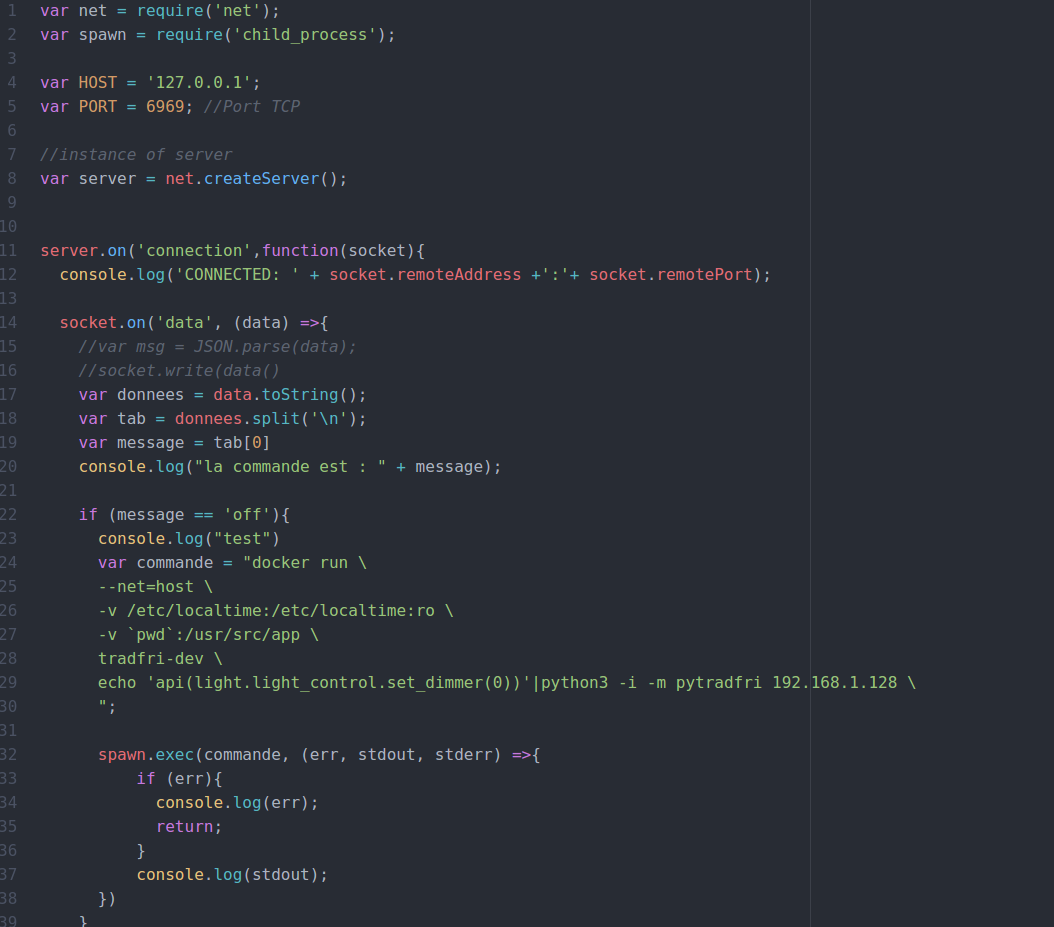


Lorsque l’utilisateur tape un mot, ce dernier va être analyser par la partie traitement du langage et va ensuite être comparé à la base de données. Avec cette requête, cela permet de voir si le verbe tapé par l’utilisateur appartient ou non à la base de données. Par exemple, s’il tape “activate the light”, le verbe *activate* appartient bien à la base de données. Ainsi, cette commande va permettre de retourner la commande qui est associé au verbe. On récupère cette commande à l’aide de la fonction “getName()“ Cette commande est ensuite envoyée au serveur.

4. Serveur (voir annexe 4)



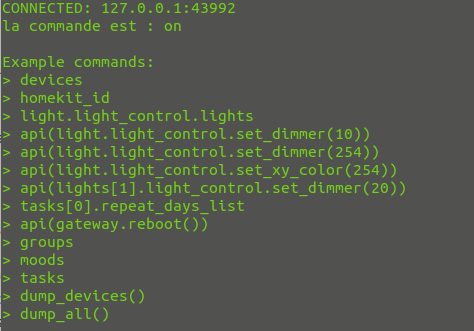
Dans cette partie, on récupère les données de la socket que l’on va split au niveau des retour à la ligne (lors de l’envoie de données par la partie traitement du langage, la commande est envoyée ainsi que de nombreux retour à la ligne).



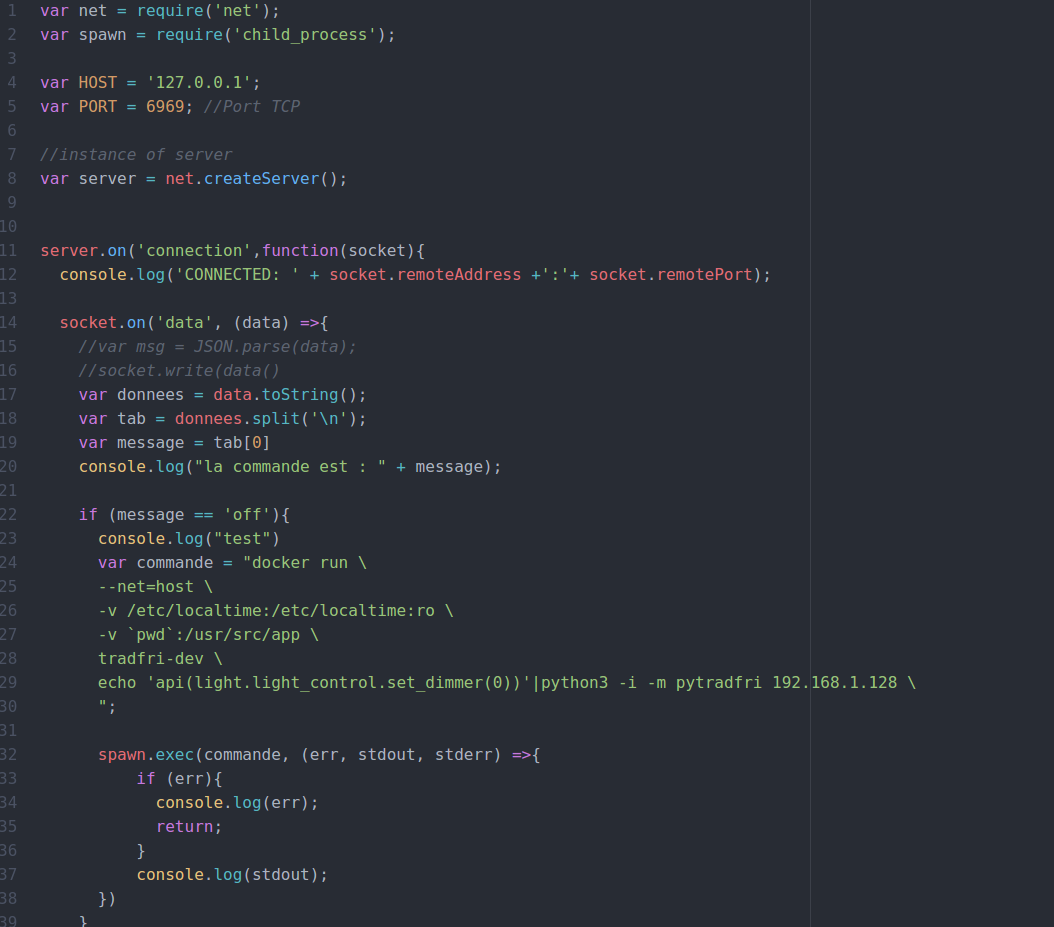
Ensuite, on teste si la commande reçue correspond à la commande “off”. Si c’est le cas, on run le docker puis on exécute la commande suivante :

*“api(light.light\_control.set\_dimmer(0)) | python3 -i -m pytradfri 192.168.1.128”*

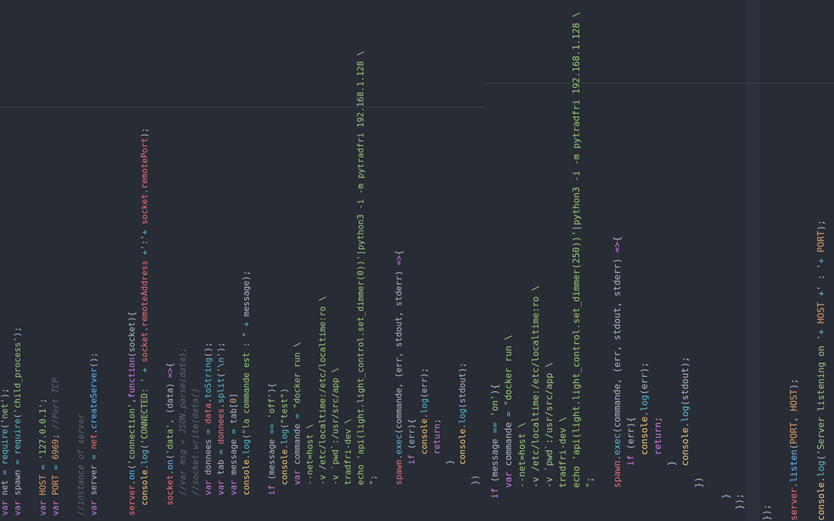
La commande *“python3 -i -m pytradfri 192.168.1.128”* permet de se connecter à la box qui possède l’adresse IP 192.168.1.128. Cela nous permet d’avoir accès aux commandes suivantes :



Ensuite la commande *“api(light.light\_control.set\_dimmer(0))”* permet de mettre la luminosité de la lumière à 0 et donc de l’éteindre.

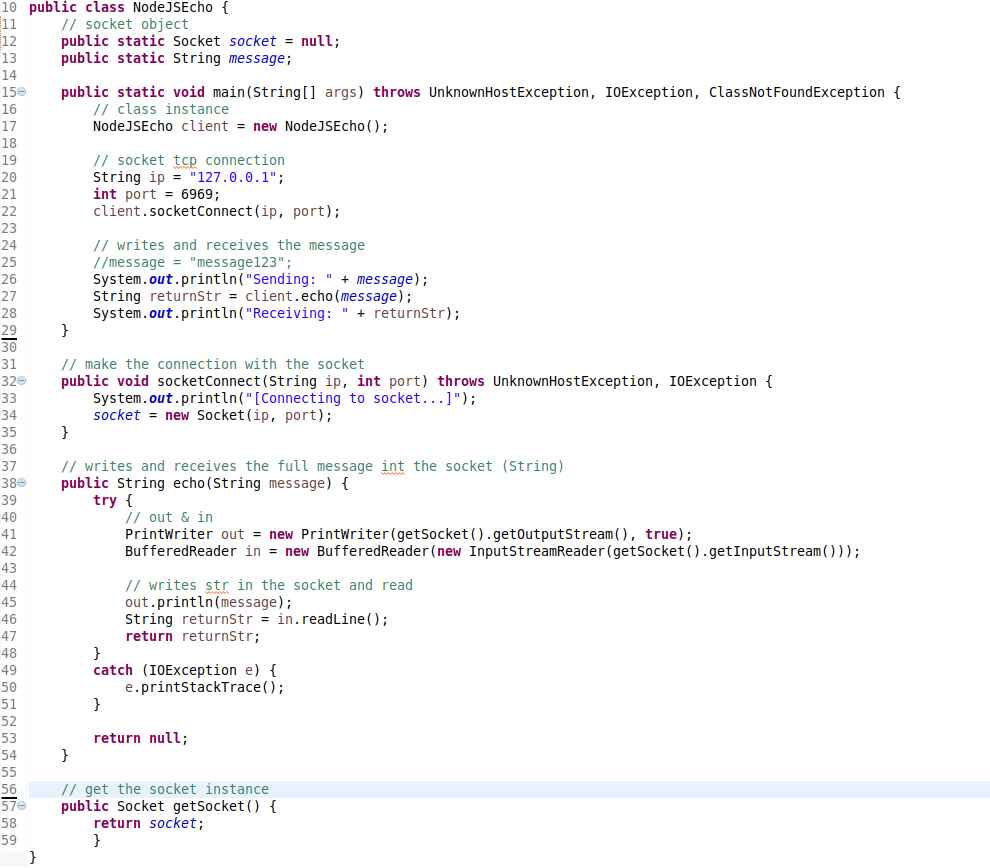


La commande *“spawn.exec()”* permet d'exécuter la commande.



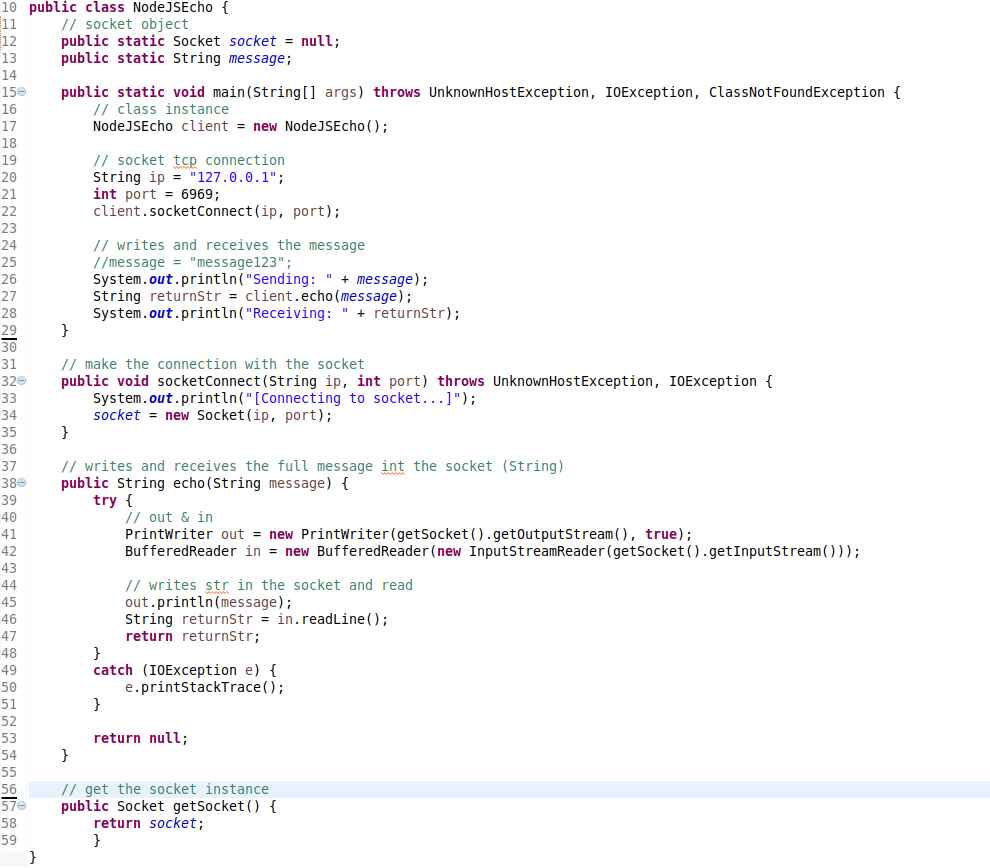
Enfin, Le serveur écoute sur le port 6969 et sur l’adresse 127.0.0.1.

5. Relation serveur et traitement du langage (voir annexe 5)



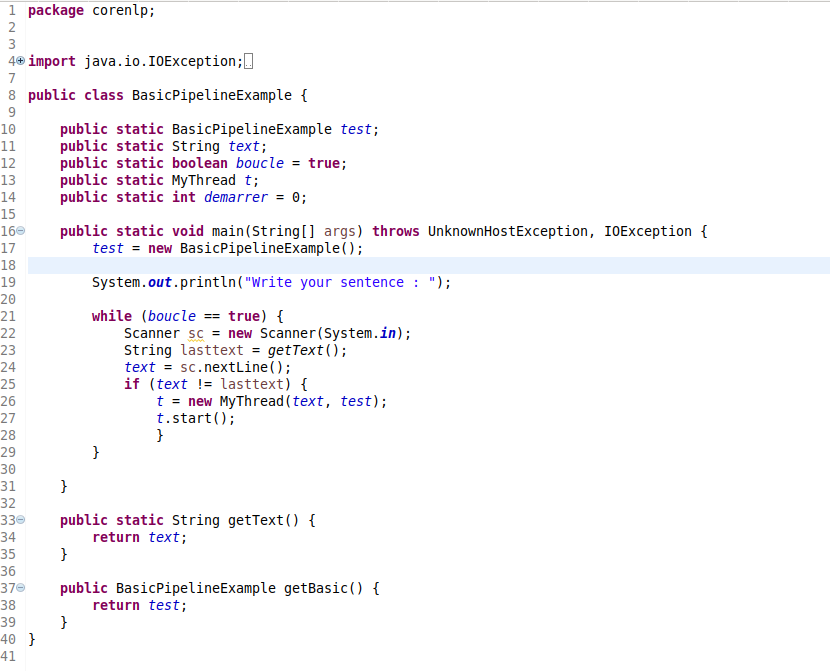
On définit ici, l’adresse IP et le port à lesquelles la socket doit être reliée. Ici 127.0.0.1 et 6969.

Ensuite, on se connecte grâce à la fonction *socketConnect().*

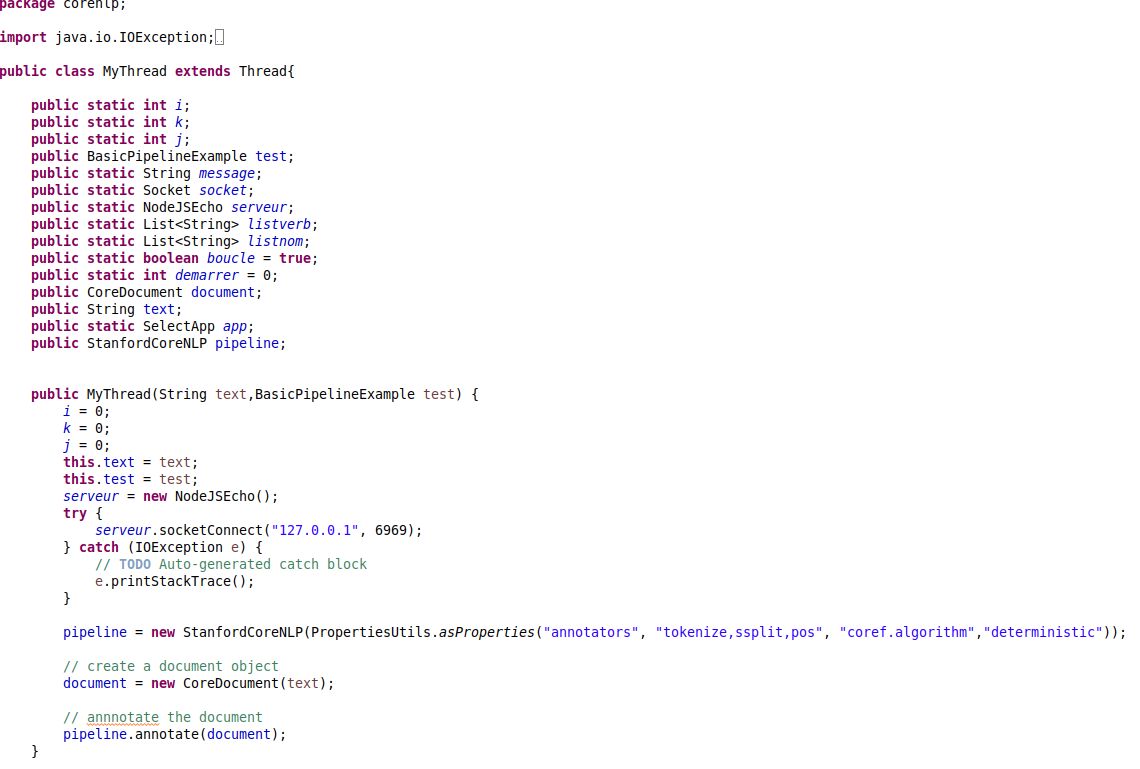


Cette fonction permet d’envoyer un message à travers la socket, dans notre projet, ce sera la commande qui sera envoyé à travers la socket. Le serveur traitera ensuite cette commande.

6. Gestion des commandes de l’utilisateur (voir annexe 6)



Ici, nous demandons à l’utilisateur d’écrire sa commande. Pour permettre à l’utilisateur de taper des commandes en boucle, nous avons créé un Thread. Dans ce Thread que nous allons développer dans la partie suivante, nous envoyons un objet de type “BasicPipelineExample” et un “String”. Ce “String” représente la commande que l’utilisateur a saisi au clavier.

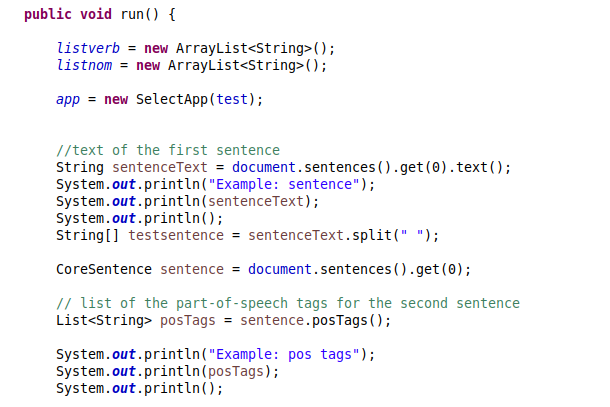
7. Thread (voir annexe 7)

Dans un premier temps, on récupère les informations envoyées par l’utilisateur précédemment.

Ensuite, on connecte la partie traitement du langage au serveur à l’aide de la socket.

Enfin, nous utilisons des fonctions présentes dans CoreNLP :

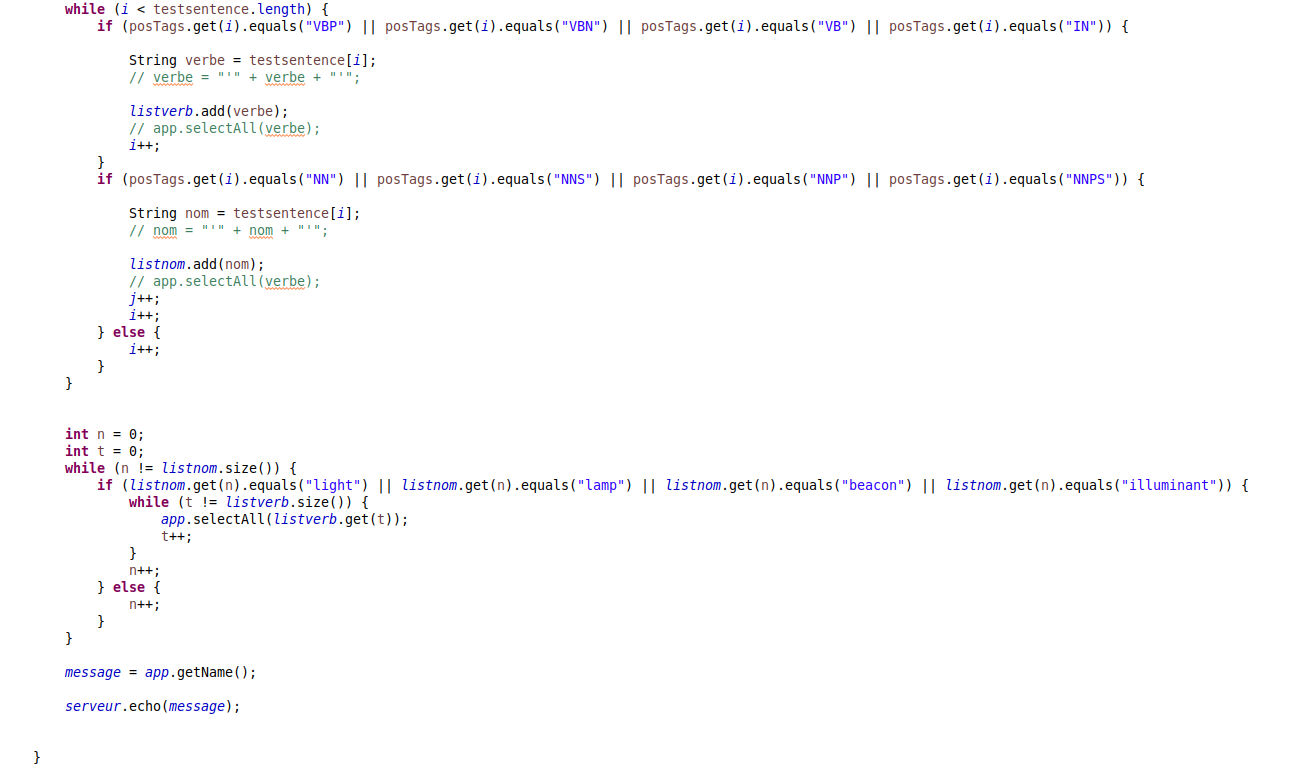
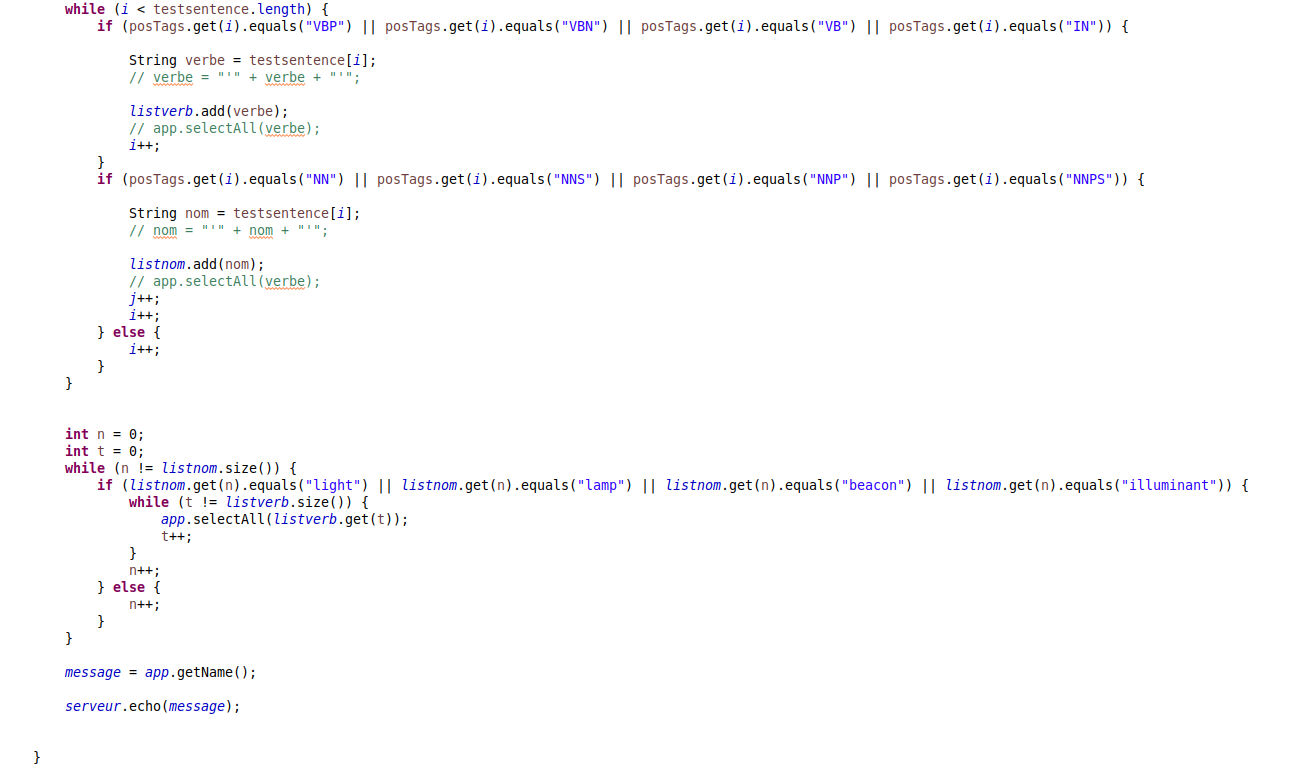
* **StanfordCoreNLP :**
  + **“annotators”** : Permet de choisir quelles annotations nous souhaitons mettre sur notre document
  + “**tokenize”** : génère les annotations
  + **“ssplit”**: Divise une séquence de jetons en phrase
  + **“pos”** : permet de ressortir le type des mots (verbe infinitif, pronoms, …)
  + **“coref.algorithm”** : Permet de choisir quel coréférence/algorithme nous voulons utiliser parmi *“neural, deterministic, statistical”*. Le deterministic étant le plus rapide, nous avons donc choisi ce dernier.
  + **Deterministic:** Résolution de coréférence rapide basée sur des règles pour l’anglais et le chinois
* **Coredocument :** Création d’un document objet
* **Annotate() :** Annotation du document créé précédemment à l’aide des annotations définis auparavant



Lorsque le Thread est lancé, on créer deux listes vides :

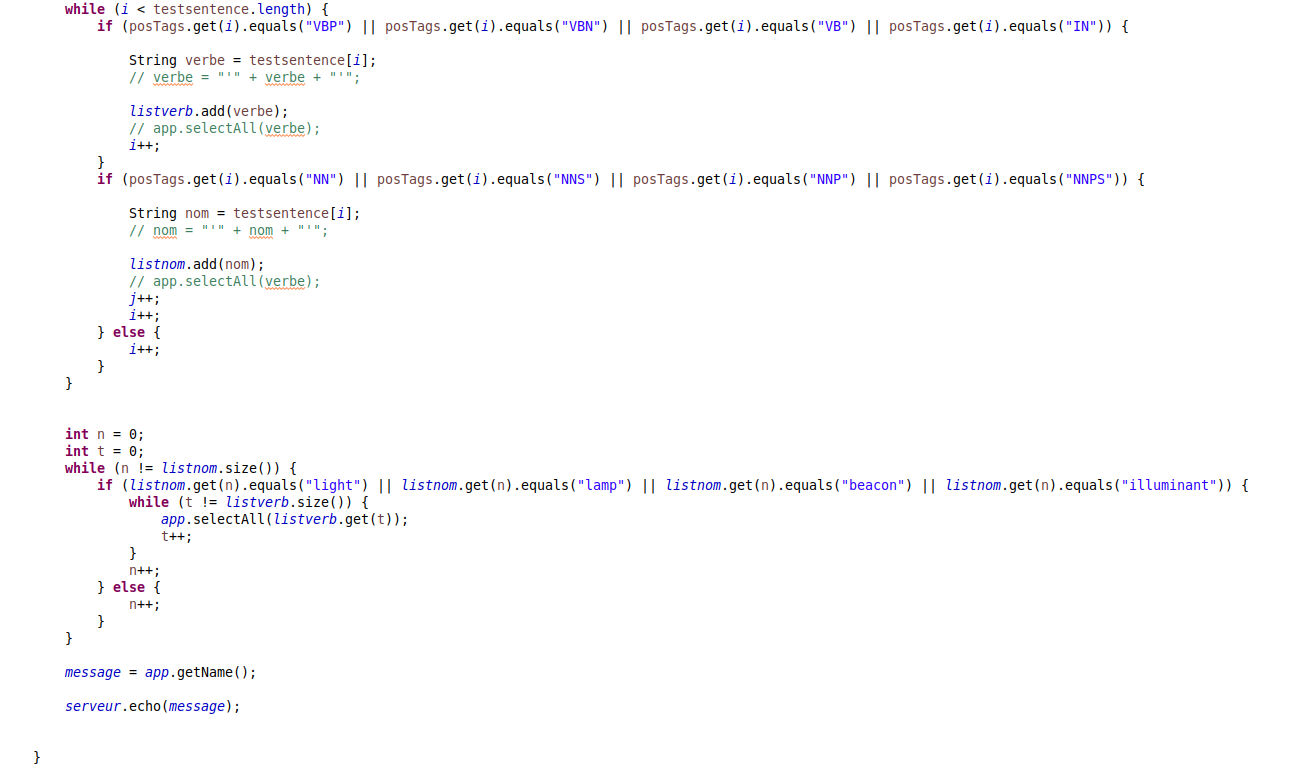
* Liste de verbe : Permet de stocker tous les verbes de la phrase
* Liste de noms : Permet de stocker tous les noms de la phrase

On stocke ensuite dans un tableau toute la phrase que l’on a découpé à chaque espace. Ici, le programme nous ressort la phrase tapée par l’utilisateur mais également la liste contenant les annotations sur les différents types des mots de la phrase.

 Dans un premier temps on parcourt le tableau avec les termes de la phrase puis on ajoute les noms dans la liste de noms et les verbes dans la liste de verbes.

Ensuite on parcourt la liste de noms, si cette dernière contient le mot “light” ou “lamp” ou “beacon” ou “illuminant” alors le verbe présent dans la phrase va être analyser. Cela permet d’éviter les phrases du type “activate the chicken”.

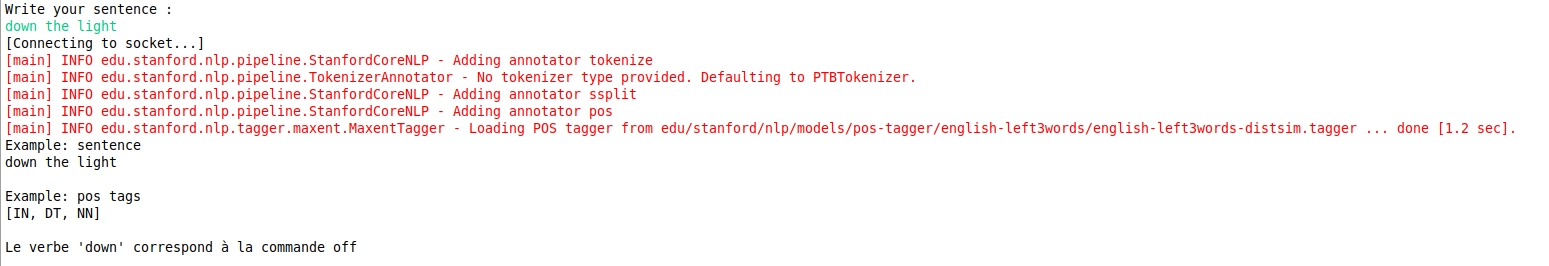
Ensuite, on exécute la requête “selectAll()” en utilisant le verbe de la phrase. Comme on l’a vu précédemment, cette requête permet d’obtenir la commande en fonction du verbe.



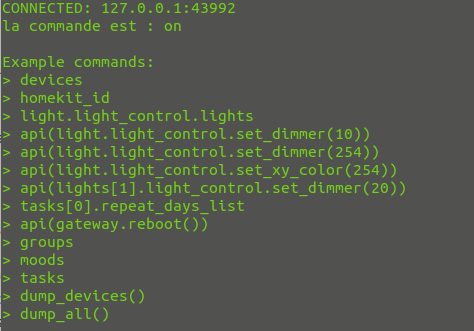
Ensuite, “getName()” permet d’obtenir la commande ressortie par la requête SQL.

On envoie ensuite la commande au serveur.

8. Résultat des commandes sur Eclipse



9. Résultat des commandes sur le terminal

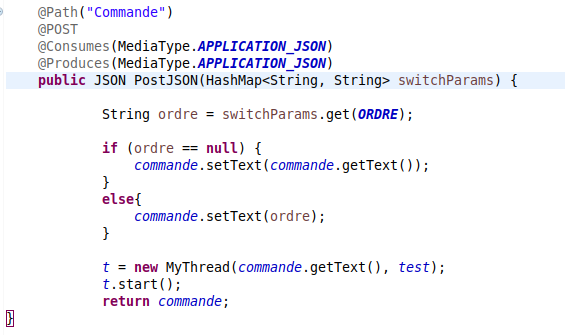


Voici les commandes que nous avons utilisé :

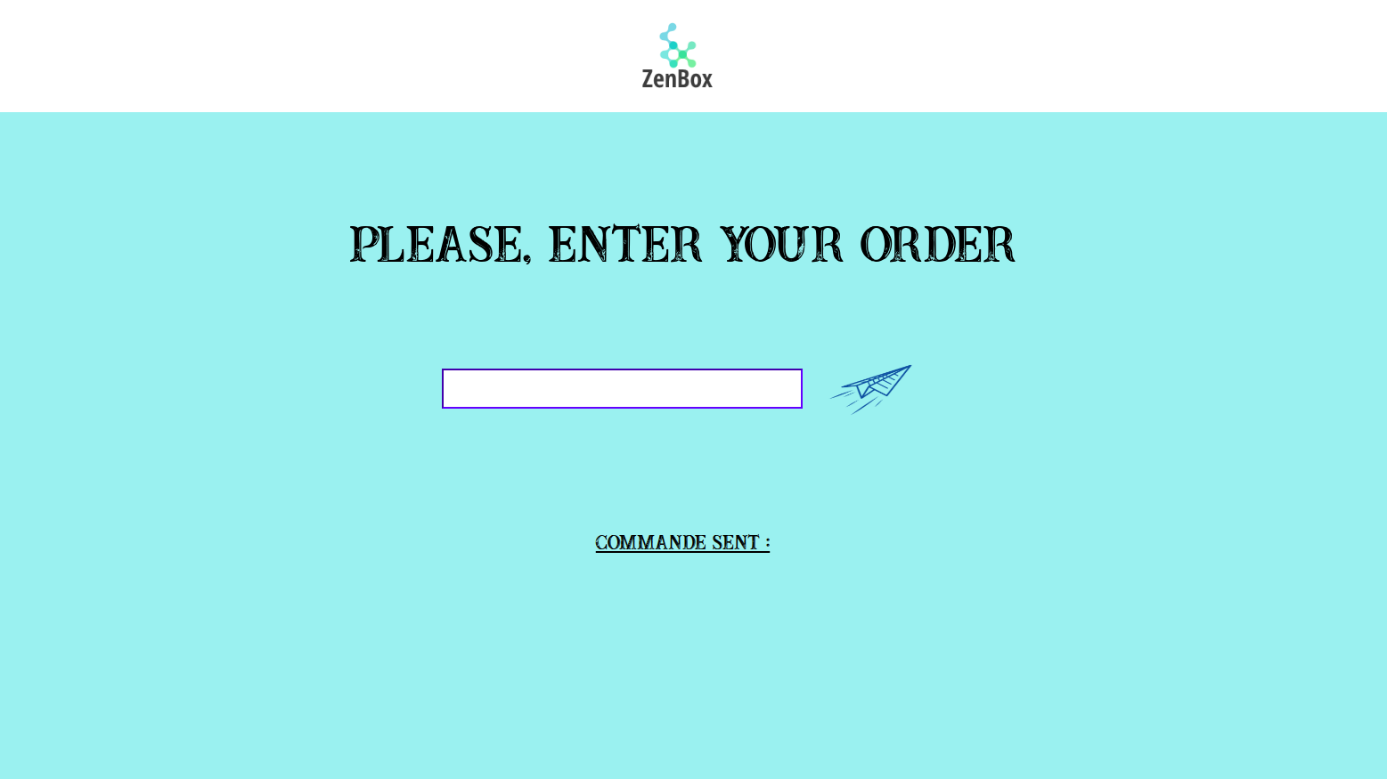
* **devices** : liste tous les objets présents sur le réseau
* **light.light\_control.lights** : donne l’état des différentes lampes ainsi que leur id
* **api(light.light\_control.set\_dimmer(250))** : Règle la luminosité de toute les lumières
* **api(light[1].light\_control.set\_dimmer(250))** : Règle la luminosité d’une lampe précise

10. Partie React

Nous avons choisi de développer notre application sous React. Ainsi cela permet à l’utilisateur d’avoir une interface pour pouvoir envoyer ses commandes. Cette partie engendre une légère modification dans le code Java :



Voici, ce que l’utilisateur obtient en lançant “npm start”



Nous fonctionnerons désormais avec l’API REST.

Voici le code exécuté lors du clique sur le bouton de la part de l’utilisateur :



Un JSON contenant le message est envoyé.

B- Difficultés rencontrés

Nous avons perdu du temps sur la partie contrôle des lampes connectées Ikea car nous sommes partis de pytradfri une librairie open source déposé par ggravlingen sur github permettant de contrôler les lampes IKEA et malgré de nombreuses heures passé à travailler sur ce programme et à comprendre son fonctionnement nous n’arrivions pas à contrôler les lampes. Que ce soit en installant le programme de la façon conseillée sur github ou encore en essayant d’envoyer uniquement les commandes libcoap à la passerelle.

Nous avons donc décidé d’utiliser l’image docker proposé sur github afin de pouvoir avancer cependant nous n’avions aucune connaissance de docker et nous avons donc dû prendre un temps pour la compréhension de son fonctionnement.

De plus nous avons également perdu du temps sur la compréhension de CoreNLP en se lançant sous nodeJs alors que son fonctionnement sous Java est plus simpliste.

Enfin, la dernière difficulté était de gérer la partie docker dans le serveur. En effet, il fallait être capable de “run” le docker dans notre serveur en nodeJs puis d'exécuter la commande souhaitée.

Pour la suite

Dans cette partie nous allons énoncés les différentes fonctionnalités que nous n’avons pas eu le temps de réaliser mais qui seraient intéressants d’implémenter dans la Zenbox. De plus, nous verrons aussi la solution que nous aurions voulu obtenir au final.

A- Travaux possible

À partir de maintenant notre but est d’améliorer encore un peu la vitesse de notre système. Pour cela, il nous faudra centraliser le serveur sur l’image docker. De cette manière, il nous sera facile de placer le docker sur une des raspberry pi tandis que sur notre second raspberry il nous suffira d’installer java et Eclipse et faire fonctionner notre partie traitement du langage. Nous aurons donc bien comme prévu deux raspberry à l’intérieur de notre ZenBox, une pour la partie serveur et communication avec les objets connectés et une pour la partie traitement du langage. C’est pourquoi nous sommes actuellement en train de nous renseigner sur la création d’image docker et la gestion des dépendances. Nous prévoyons de faire tourner l’image Docker car elle est légère et permet une adaptation plus facile d’une machine à une autre. Les premiers tests seront réalisés lors de la prochaine séance.

Tout au long du projet nous mettons à jour nos avancement sur le site et c’est donc toujours un travail un cours, cependant il est toujours en local mais nous pensons le mettre en ligne assez rapidement, nous attendions d’avoir un début de solution pour que le site ait un plus grand intérêt.

Une fois que nous aurons fini cela, il nous faudra renforcer le système de traitement de langage. Pour ce faire nous comptons continuer à l’entraîner. Nous allons travailler à répondre à des structures de phrases de plus en plus complexes jusqu’à arriver à un système fiable qui comprendrait une grande partie des phrases écrites par l’utilisateur.

Nous allons également travailler à différencier quelles lampes nous allons commander. Nous avons en notre possession deux lampes ikea et nous pourrons donc améliorer notre système pour qu’il puisse gérer plusieurs équipements en même temps. Afin de les différencier, il faudra d’abord demander à l’utilisateur de les différencier en fonction du lieu dans lequel il les place, d’un identifiant qu’il voudrait utiliser pour les nommer etc. Ce mode de différentiation sera stocké à côté dans une base de données. Ainsi il sera possible au client de définir plusieurs types d’appellations qui seront enregistrées par notre solution afin de commander l’équipement souhaité.

Enfin nous allons mettre notre système sur une carte raspberry. Cette étape ne devrait pas être compliquée, sauf si la carte n’est pas assez puissante pour faire tourner notre système. Dans ce cas nous d’utiliser des cartes plus puissantes. Dans la même optique nous allons développer notre boitier avec le fabLab.

Il nous restera ensuite à développer l’application mobile nous permettant d’utiliser notre système sur mobile.

B- Solution finale espérée

Notre but est d’arriver à une solution déployée en tant qu’application mobile. Il sera possible pour l’utilisateur de commander quel type de lampe il veut allumer ou éteindre, et ce avec une grande variété de phrases. Les erreurs de compréhension devront être réduites au possible afin qu’il soit agréable d’utiliser notre système. Les commandes seront prises en compte dans un bref délai. Physiquement, notre solution sera dans un boitier comme prévu initialement.

Apport du projet

Ce projet nous a permis de découvrir différentes techniques de travail. En effet, au départ du projet, nous étions mal organisés et nous nous sommes aperçus que le fait de ne pas définir des objectifs en début de cours ne nous permettait pas d’avancer convenablement. Ainsi lorsque l’on travaille à plusieurs il faut que le projet soit bien dirigé avec des deadlines pour chaque objectif.

Nous avons pu voir une application concrète de certains cours que nous avons au sein de notre formation et nous avons aussi appris à chercher des solutions par nous-même vis à vis des différents problèmes que nous avons rencontrés que ce soit en consultant les documentations ou en cherchant des réponses sur stack overflow.

Concernant les délivrables, cela nous a permis de synthétiser notre avancement. Ceci nous sera utile pour nos futurs rendus en entreprise. En effet, nous devions les rédiger de manière professionnelle et très explicite.

De plus, nous avons pu découvrir le système de machine learning, le logiciel CoreNLP et l’utilisation d’image docker qui est très intéressante par rapport à l’utilisation de machines virtuelles.

Enfin, ce projet aura été le premier projet technique complet qu’on a pu développer et auquel on a apporté une solution certes incomplète mais tout de même fonctionnel.

Pour résumer, nous avons pu nous rendre compte de l’importance du management de projet mais nous avons aussi pu acquérir des compétences techniques.

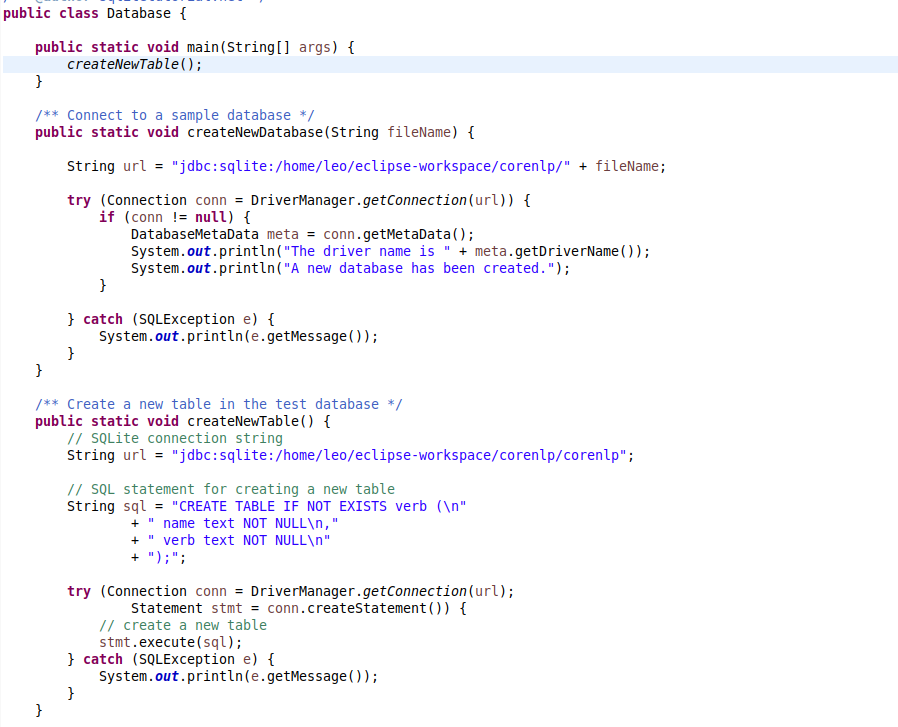
Conclusion

Nous avons atteint nos objectifs principaux durant ce projet. Notre solution est fonctionnelle et nous pouvons bien communiquer avec notre application en langage naturel que notre système analyse avant de pouvoir allumer ou éteindre une lampe. Plusieurs choses nous ont ralenti durant ce projet. Nous avons passé beaucoup plus de temps que prévu à comprendre le fonctionnement de CoreNLP et nous ne l’avons même pas utilisé à son plein potentiel. Tout ce temps perdu était crucial car il nous a empêché d’avancer, malgré cela nous sommes montés en compétences dans le domaine des systèmes de traitement de langage. D’autres choses nous ont empêché d’aller plus loin, comme l’indisponibilité d’autres équipements connectés, nous avons donc dû en rester à contrôler seulement une lampe de la marque IKEA.

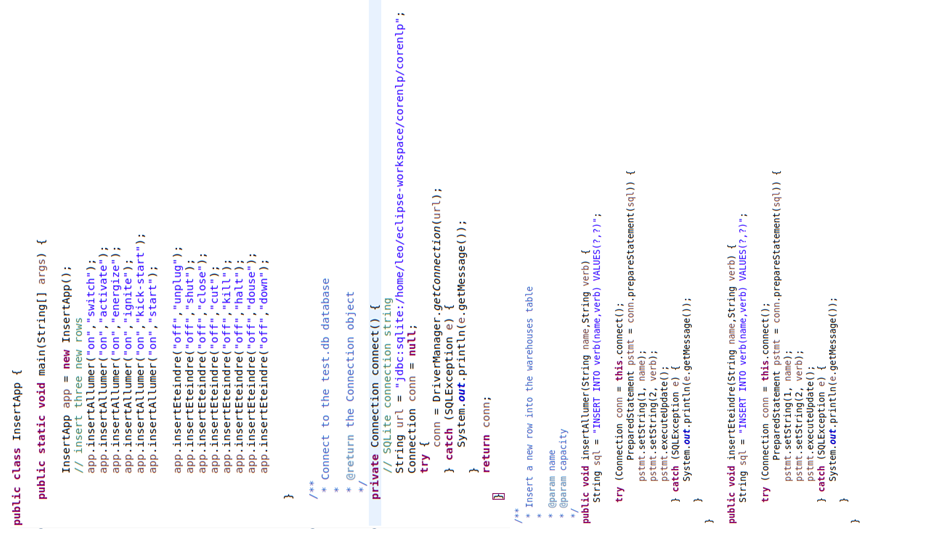
Ce qui nous a le plus retardé dans l’avancement de notre projet était notre vision trop floue du système de traitement de langage et de son fonctionnement global. Même si nous pensions avoir clairement compris comment tout cela fonctionnait, il n’en était rien et nous avons dû faire face à de nombreux problèmes que l’on a mis du temps à surmonter à chaque fois. Il est certain que nous avions un manque de compétences dans ce domaine et il nous a fallu du temps pour le combler.

Annexes

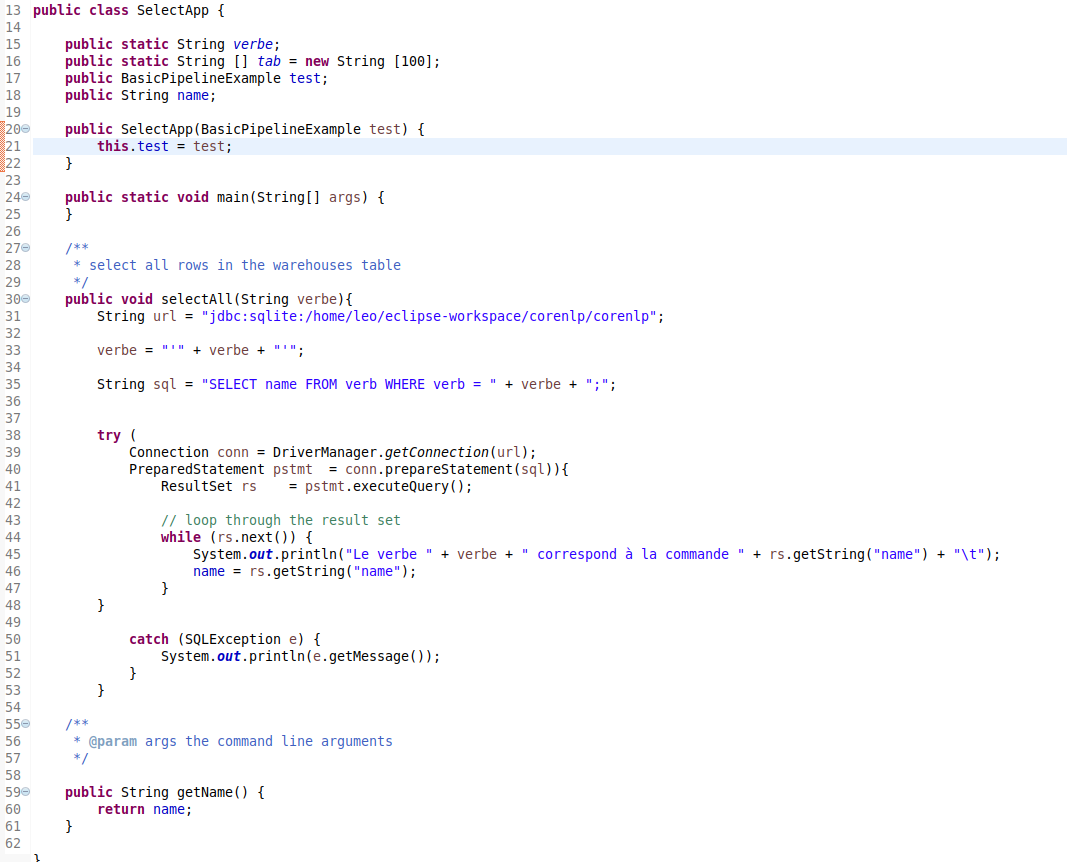
Annexe 1 : Création de la base de données

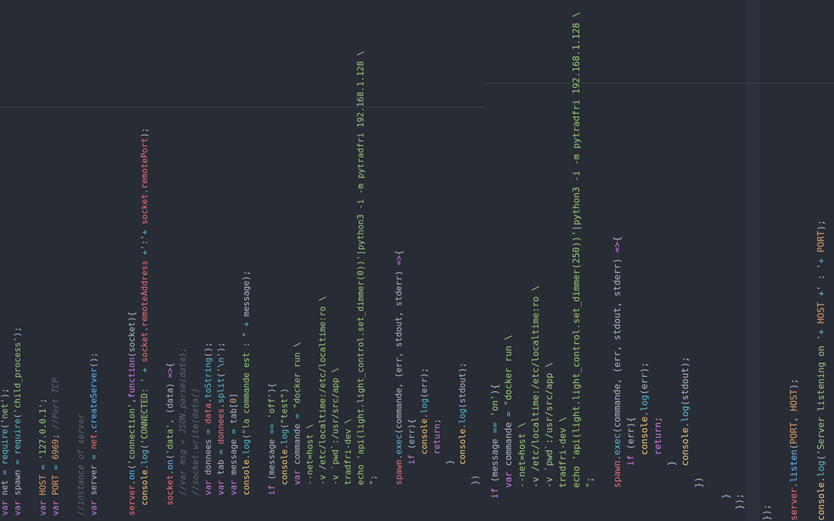


Annexe 2 : Insertion et connexion à la base de données

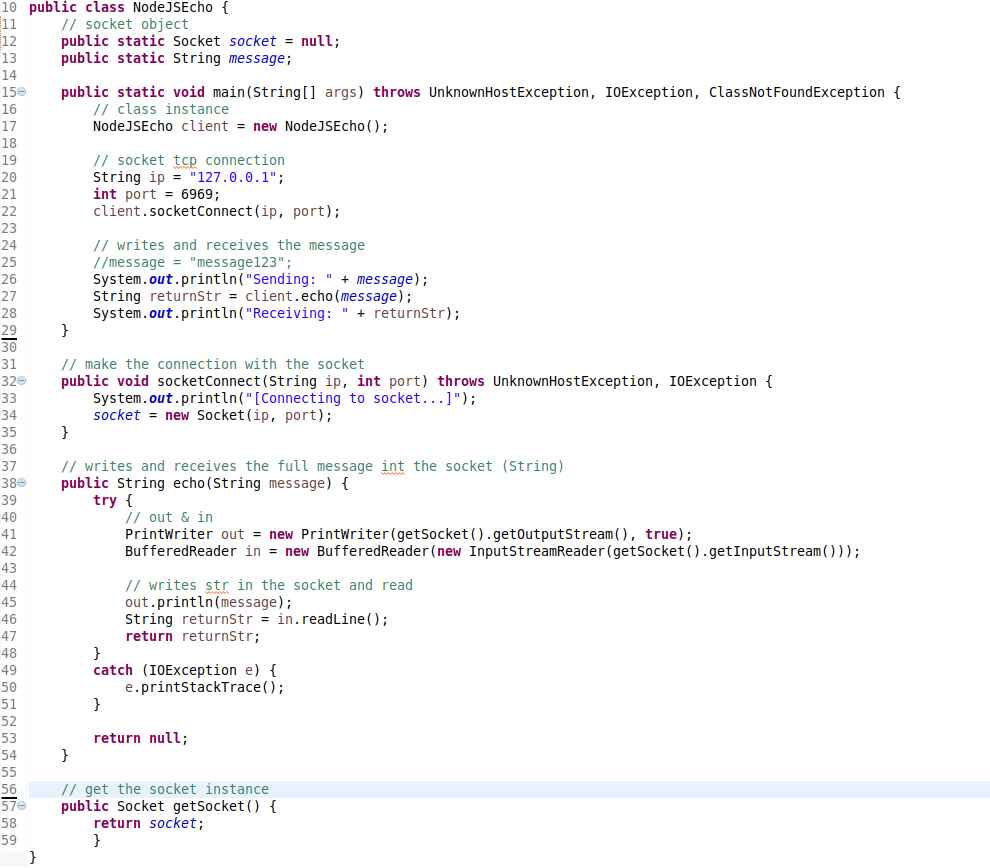


Annexe 3 : Sélection des données dans la base

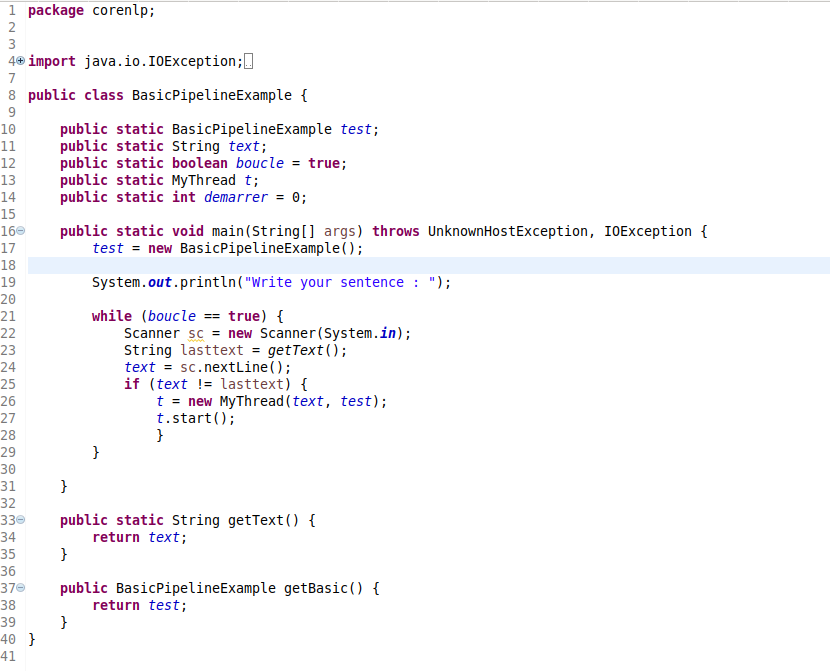


Annexe 4 : Serveur

Annexe 5 : Relation serveur et traitement du langage



Annexe 6 : Gestion des commandes de l’utilisateur



Annexe 7 : Thread



