UNIVERSITÉ DE RENNES 1

ESIR ECOLE SUPERIEURE D'INGENIEURS DE RENNES

27/04/2018

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 »

Aguirre Max & Izabelle François & Guilpain Léo

Table des matières

Introduction 2
Rappel du projet 2
A- Schéma Global (+ quelques explications voir Délivrable 2) 2
B- Gantt 2
C- Les Étapes de notre projet 3
Rapport d'avancement 4
A- Explication des réalisations 4
1. Création de la base de données (voir annexe 1) 4
2. Insertion et connexion à la base de données (voir annexe 2) 5
3. Sélection des données dans la base (voir annexe 3) 6
4. Serveur (voir annexe 4) 7
5. Relation serveur et traitement du langage (voir annexe 5) 8
6. Gestion des commandes de l'utilisateur (voir annexe 6) 9
7. Thread (voir annexe 7) 9
8. Résultat des commandes sur Eclipse 12
8. Résultat des commandes sur le terminal 12
B- Difficultés rencontrés 12
Pour la suite 13
A- Travaux en cours 13
B- Travaux à venir 13
C- Solution finale espérée (réaliste) 13
Conclusion 14
Annexes 15
Annexe 1 : Création de la base de données 15
Annexe 2 : Insertion et connexion à la base de données 16
Annexe 3 : Sélection des données dans la base 17
Annexe 4 : Serveur 18
Annexe 5 : Relation serveur et traitement du langage 19
Annexe 6 : Gestion des commandes de l'utilisateur 20
Annexe 7 : Thread 21

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. Enfin on terminera ce délivrable en décrivant ce que nous comptons faire par la suite, et en proposant une estimation réaliste de ce à quoi nous pensons pouvoir arriver d'ici la fin de cette année.

Rappel du projet

A- Schéma Global (+ quelques explications voir Délivrable 2)

B- Gantt

Nom	Date de début	Date de fin	Durée
Projet Boitier ZenBox	03/10/17	04/12/17	45
Soutenance n°1	01/12/17	08/12/17	6
Journée Porte Ouverte	26/01/18	03/02/18	6
Partie Recherche et Formation	03/10/17	17/11/17	34
Etat de l'art	03/10/17	13/10/17	9
Recherches sur les différents protocoles de transmission	03/10/17	13/10/17	9
Formation sur les fonctionnalités de base du Raspberry pi	03/10/17	13/10/17	9
Affectation des tâches	03/10/17	13/10/17	9
Détermination des équipements nécessaires	13/10/17	17/11/17	26
Partie Communication	03/10/17	25/05/18	169
Délivrable n°1	03/10/17	13/10/17	9
Rassembler les informations	03/10/17	13/10/17	9
Mise en page + Webographie	03/10/17	13/10/17	9
Délivrable n°2	13/10/17	17/11/17	26
Délivrable n°3	17/11/17	08/12/17	16
Mise en ligne d'un site	24/11/17	25/05/18	131
Création du site	24/11/17	25/05/18	131
Mise à jour de l'avancement du projet	24/11/17	25/05/18	131
Partie Boitier + Tchat Box	24/11/17	25/05/18	131
Partie Traitement du langage	24/11/17	25/05/18	131
Analyser une demande et la renvoyer sous le bon format	24/11/17	08/01/18	32
Entrainer le logiciel	08/01/18	25/05/18	100
Partie Serveur	24/11/17	08/01/18	32
Authentification	01/12/17	08/01/18	27
Transfert des messages reçus par la partie traitement de langage	24/11/17	08/01/18	32
Partie Client	24/11/17	25/05/18	131
Solution test (cf délivrable 2)	24/11/17	08/01/18	32
Application Android	08/01/18	25/05/18	100
Création du boîtier	08/01/18	25/05/18	100

C- 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 qu'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. A 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 phrases typiques ou expressions qu'un utilisateur pouvait utiliser pour parler à notre système. En même temps, nous avons développé le côté serveur de notre solution et nous sommes arrivés à une première version dans laquelle nous arrivions à retourner les commandes appropriées à la partie serveur à partir d'une commande utilisateur.

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. **Expliquer ce que vous avez fait du coup**

Rapport d'avancement

A- Explication des réalisations

Les codes seront disponibles dans leur intégralité en annexe.

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

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

```
/** Connect to a sample database */
public static void createNewDatabase(String fileName) {

    String url = "jdbc:sqlite:/home/leo/eclipse-workspace/corenlp/" + fileName;

    try (Connection conn = DriverManager.getConnection(url)) {
        if (conn != null) {
            DatabaseMetaData meta = conn.getMetaData();
            System.out.println("The driver name is " + meta.getDriverName());
            System.out.println("A new database has been created.");
        }

    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
}
```

Dans ce code, nous avons créé la base de données que l'on a nommé "corenlp".

Ensuite, nous avons créé une table :

```
/** Create a new table in the test database */
public static void createNewTable() {
    // SQLite connection string
    String url = "jdbc:sqlite:/home/leo/eclipse-workspace/corenlp/corenlp";

    // SQL statement for creating a new table
    String sql = "CREATE TABLE IF NOT EXISTS verb (\n"
            + " name text NOT NULL\n,"
            + " verb text NOT NULL\n"
            + ");";

    try (Connection conn = DriverManager.getConnection(url);
            Statement stmt = conn.createStatement()) {
        // create a new table
        stmt.execute(sql);
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
}
```

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)

```java
    /**
     * Connect to the test.db database
     *
     * @return the Connection object
     */
    private Connection connect() {
        // SQLite connection string
        String url = "jdbc:sqlite:/home/leo/eclipse-workspace/corenlp/corenlp";
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(url);
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
        return conn;
    }
```

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

```java
/**
 * Insert a new row into the warehouses table
 *
 * @param name
 * @param capacity
 */
public void insertAllumer(String name,String verb) {
    String sql = "INSERT INTO verb(name,verb) VALUES(?,?)";

    try (Connection conn = this.connect();
            PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, name);
        pstmt.setString(2, verb);
        pstmt.executeUpdate();
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
}

public void insertEteindre(String name,String verb) {
    String sql = "INSERT INTO verb(name,verb) VALUES(?,?)";

    try (Connection conn = this.connect();
            PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, name);
        pstmt.setString(2, verb);
        pstmt.executeUpdate();
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
}
```

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

```
        app.insertAllumer("on","switch");
        app.insertAllumer("on","activate");
        app.insertAllumer("on","energize");
        app.insertAllumer("on","ignite");
        app.insertAllumer("on","kick-start");
        app.insertAllumer("on","start");

        app.insertEteindre("off","unplug");
        app.insertEteindre("off","shut");
        app.insertEteindre("off","close");
        app.insertEteindre("off","cut");
        app.insertEteindre("off","kill");
        app.insertEteindre("off","halt");
        app.insertEteindre("off","douse");
        app.insertEteindre("off","down");

    }
```

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)

```
29      */
30      public void selectAll(String verbe){
31          String url = "jdbc:sqlite:/home/leo/eclipse-workspace/corenlp/corenlp";
32
33          verbe = "'" + verbe + "'";
34
35          String sql = "SELECT name FROM verb WHERE verb = " + verbe + ";";
36
37
38          try (
39              Connection conn = DriverManager.getConnection(url);
40              PreparedStatement pstmt  = conn.prepareStatement(sql)){
41                  ResultSet rs    = pstmt.executeQuery();
42
43                  // loop through the result set
44                  while (rs.next()) {
45                      System.out.println("Le verbe " + verbe + " correspond à la commande " + rs.getString("name") + "\t");
46                      name = rs.getString("name");
47                  }
48          }
49
50              catch (SQLException e) {
51                  System.out.println(e.getMessage());
52              }
53          }
54
55      /**
56       * @param args the command line arguments
57       */
58
59      public String getName() {
60          return name;
61      }
62
}
```

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)

```
socket.on('data', (data) =>{
  //var msg = JSON.parse(data);
  //socket.write(data()
  var donnees = data.toString();
  var tab = donnees.split('\n');
  var message = tab[0]
  console.log("la commande est : " + message);
```

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).

```
if (message == 'off'){
  console.log("test")
  var commande = "docker run \
  --net=host \
  -v /etc/localtime:/etc/localtime:ro \
  -v `pwd`:/usr/src/app \
  tradfri-dev \
  echo 'api(light.light_control.set_dimmer(0))'|python3 -i -m pytradfri 192.168.1.128 \
  ";
```

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 :

```
Example commands:
> devices
> homekit_id
> light.light_control.lights
> api(light.light_control.set_dimmer(10))
> api(light.light_control.set_dimmer(254))
> api(light.light_control.set_xy_color(254))
> api(lights[1].light_control.set_dimmer(20))
> tasks[0].repeat_days_list
> api(gateway.reboot())
> groups
> moods
> tasks
> dump_devices()
> dump_all()
```

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.

```
        spawn.exec(commande, (err, stdout, stderr) =>{
            if (err){
              console.log(err);
              return;
            }
            console.log(stdout);
        })
    }
```

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

```
server.listen(PORT, HOST);

console.log('Server listening on '+ HOST +' : '+ PORT);
```

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)

```
19          // socket tcp connection
20          String ip = "127.0.0.1";
21          int port = 6969;
22          client.socketConnect(ip, port);
23
```

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()*.

```
37      // writes and receives the full message int the socket (String)
38      public String echo(String message) {
39          try {
40              // out & in
41              PrintWriter out = new PrintWriter(getSocket().getOutputStream(), true);
42              BufferedReader in = new BufferedReader(new InputStreamReader(getSocket().getInputStream()));
43
44              // writes str in the socket and read
45              out.println(message);
46              String returnStr = in.readLine();
47              return returnStr;
48          }
49          catch (IOException e) {
50              e.printStackTrace();
51          }
52
53          return null;
54      }
```

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)

```
19          System.out.println("Write your sentence : ");
20
21          while (boucle == true) {
22              Scanner sc = new Scanner(System.in);
23              String lasttext = getText();
24              text = sc.nextLine();
25              if (text != lasttext) {
26                  t = new MyThread(text, test);
27                  t.start();
28                  }
29          }
30
31      }
```

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)

```
public MyThread(String text,BasicPipelineExample test) {
    i = 0;
    k = 0;
    j = 0;
    this.text = text;
    this.test = test;
    serveur = new NodeJSEcho();
    try {
        serveur.socketConnect("127.0.0.1", 6969);
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    pipeline = new StanfordCoreNLP(PropertiesUtils.asProperties("annotators", "tokenize,ssplit,pos", "coref.algorithm","deterministic"));

    // create a document object
    document = new CoreDocument(text);

    // annnotate the document
    pipeline.annotate(document);
}
```

Dans un premier temps, on récupère les informations envoyés 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 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

```
public void run() {

    listverb = new ArrayList<String>();
    listnom = new ArrayList<String>();

    app = new SelectApp(test);


    //text of the first sentence
    String sentenceText = document.sentences().get(0).text();
    System.out.println("Example: sentence");
    System.out.println(sentenceText);
    System.out.println();
    String[] testsentence = sentenceText.split(" ");

    CoreSentence sentence = document.sentences().get(0);

    // list of the part-of-speech tags for the second sentence
    List<String> posTags = sentence.posTags();

    System.out.println("Example: pos tags");
    System.out.println(posTags);
    System.out.println();
```

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.

```
while (i < testsentence.length) {
    if (posTags.get(i).equals("VBP") || posTags.get(i).equals("VBN") || posTags.get(i).equals("VB") || posTags.get(i).equals("IN")) {

        String verbe = testsentence[i];
        // verbe = "'" + verbe + "'";

        listverb.add(verbe);
        // app.selectAll(verbe);
        i++;
    }
    if (posTags.get(i).equals("NN") || posTags.get(i).equals("NNS") || posTags.get(i).equals("NNP") || posTags.get(i).equals("NNPS")) {

        String nom = testsentence[i];
        // nom = "'" + nom + "'";

        listnom.add(nom);
        // app.selectAll(verbe);
        j++;
        i++;
    } else {
        i++;
    }
}
```

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.

```
int n = 0;
int t = 0;
while (n != listnom.size()) {
    if (listnom.get(n).equals("light") || listnom.get(n).equals("lamp") || listnom.get(n).equals("beacon") || listnom.get(n).equals("illuminant")) {
        while (t != listverb.size()) {
            app.selectAll(listverb.get(t));
            t++;
        }
        n++;
    } else {
        n++;
    }
}
```

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.

```
message = app.getName();

serveur.echo(message);
```

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

```
Write your sentence :
down the light
[Connecting to socket...]
[main] INFO edu.stanford.nlp.pipeline.StanfordCoreNLP - Adding annotator tokenize
[main] INFO edu.stanford.nlp.pipeline.TokenizerAnnotator - No tokenizer type provided. Defaulting to PTBTokenizer.
[main] INFO edu.stanford.nlp.pipeline.StanfordCoreNLP - Adding annotator ssplit
[main] INFO edu.stanford.nlp.pipeline.StanfordCoreNLP - Adding annotator pos
[main] INFO edu.stanford.nlp.tagger.maxent.MaxentTagger - Loading POS tagger from edu/stanford/nlp/models/pos-tagger/english-left3words/english-left3words-distsim.tagger ... done [1.2 sec].
Example: sentence
down the light

Example: pos tags
[IN, DT, NN]

Le verbe 'down' correspond à la commande off
```

8. Résultat des commandes sur le terminal

```
CONNECTED: 127.0.0.1:43992
la commande est : on

Example commands:
> devices
> homekit_id
> light.light_control.lights
> api(light.light_control.set_dimmer(10))
> api(light.light_control.set_dimmer(254))
> api(light.light_control.set_xy_color(254))
> api(lights[1].light_control.set_dimmer(20))
> tasks[0].repeat_days_list
> api(gateway.reboot())
> groups
> moods
> tasks
> dump_devices()
> dump_all()
```

B- Difficultés rencontrés

Nous avons perdu du temps sur la partie contrôle des lampes connectées Ikea car nous sommes partis de pytradfri un programme open source déposé par ggravlingen sur github 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 par 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.

Pour la suite

A- Travaux en cours

À partir de maintenant notre but est d'améliorer encore un peu la vitesse de notre système. Il nous faudra également centraliser tout notre code et un des moyens serait de transférer tout notre code sur l'image docker. De cette manière, il nous suffirait donc de démarrer notre Docker et de faire tourner notre système directement dessus. C'est sur quoi nous sommes actuellement en train de travailler.

B- Travaux à venir

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. On va 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. Pour ça nous allons intégrer des analyseurs de phrases disponibles par CoreNLP plus complexes que ceux utilisés actuellement.

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 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 envisagerons de faire tourner notre système sur plusieurs cartes ou alors 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.

C- Solution finale espérée (réaliste)

Notre but est d'arriver à une solution déployée en tant qu'application mobile. Il sera possible pour l'utilisateur de commander quelle 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.

Conclusion

Annexes

Annexe 1 : Création de la base de données

```
public class Database {

    public static void main(String[] args) {
        createNewTable();
    }

    /** Connect to a sample database */
    public static void createNewDatabase(String fileName) {

        String url = "jdbc:sqlite:/home/leo/eclipse-workspace/corenlp/" + fileName;

        try (Connection conn = DriverManager.getConnection(url)) {
            if (conn != null) {
                DatabaseMetaData meta = conn.getMetaData();
                System.out.println("The driver name is " + meta.getDriverName());
                System.out.println("A new database has been created.");
            }

        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
    }

    /** Create a new table in the test database */
    public static void createNewTable() {
        // SQLite connection string
        String url = "jdbc:sqlite:/home/leo/eclipse-workspace/corenlp/corenlp";

        // SQL statement for creating a new table
        String sql = "CREATE TABLE IF NOT EXISTS verb (\n"
                + " name text NOT NULL\n,"
                + " verb text NOT NULL\n"
                + ");";

        try (Connection conn = DriverManager.getConnection(url);
                Statement stmt = conn.createStatement()) {
            // create a new table
            stmt.execute(sql);
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
    }
```

Annexe 2 : Insertion et connexion à la base de données

```
public class InsertApp {

    public static void main(String[] args) {

        InsertApp app = new InsertApp();
        // insert three new rows
        app.insertAllumer("on","switch");
        app.insertAllumer("on","activate");
        app.insertAllumer("on","energize");
        app.insertAllumer("on","ignite");
        app.insertAllumer("on","kick-start");
        app.insertAllumer("on","start");

        app.insertEteindre("off","unplug");
        app.insertEteindre("off","shut");
        app.insertEteindre("off","close");
        app.insertEteindre("off","cut");
        app.insertEteindre("off","kill");
        app.insertEteindre("off","halt");
        app.insertEteindre("off","douse");
        app.insertEteindre("off","down");

    }

    /**
     * Connect to the test.db database
     *
     * @return the Connection object
     */
    private Connection connect() {
        // SQLite connection string
        String url = "jdbc:sqlite:/home/leo/eclipse-workspace/corenlp/corenlp";
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(url);
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
        return conn;
    }
/**
 * Insert a new row into the warehouses table
 *
 * @param name
 * @param capacity
 */
public void insertAllumer(String name,String verb) {
    String sql = "INSERT INTO verb(name,verb) VALUES(?,?)";

    try (Connection conn = this.connect();
            PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, name);
        pstmt.setString(2, verb);
        pstmt.executeUpdate();
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
}

public void insertEteindre(String name,String verb) {
    String sql = "INSERT INTO verb(name,verb) VALUES(?,?)";

    try (Connection conn = this.connect();
            PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, name);
        pstmt.setString(2, verb);
        pstmt.executeUpdate();
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
}
```

Annexe 3 : Sélection des données dans la base

```java
public class SelectApp {

    public static String verbe;
    public static String [] tab = new String [100];
    public BasicPipelineExample test;
    public String name;

    public SelectApp(BasicPipelineExample test) {
        this.test = test;
    }

    public static void main(String[] args) {
    }

    /**
     * select all rows in the warehouses table
     */
    public void selectAll(String verbe){
        String url = "jdbc:sqlite:/home/leo/eclipse-workspace/corenlp/corenlp";

        verbe = "'" + verbe + "'";

        String sql = "SELECT name FROM verb WHERE verb = " + verbe + ";";


        try (
            Connection conn = DriverManager.getConnection(url);
            PreparedStatement pstmt  = conn.prepareStatement(sql)){
                ResultSet rs    = pstmt.executeQuery();

                // loop through the result set
                while (rs.next()) {
                    System.out.println("Le verbe " + verbe + " correspond à la commande " + rs.getString("name") + "\t");
                    name = rs.getString("name");
                }
        }

            catch (SQLException e) {
                System.out.println(e.getMessage());
            }
        }

    /**
     * @param args the command line arguments
     */

    public String getName() {
        return name;
    }

}
```

Annexe 4 : Serveur

```
var net = require('net');
var spawn = require('child_process');

var HOST = '127.0.0.1';
var PORT = 6969; //Port TCP

//instance of server
var server = net.createServer();


server.on('connection',function(socket){
  console.log('CONNECTED: ' + socket.remoteAddress +':'+ socket.remotePort);

  socket.on('data', (data) =>{
    //var msg = JSON.parse(data);
    //socket.write(data)
    var donnees = data.toString();
    var tab = donnees.split('\n');
    var message = tab[0]
    console.log("la commande est : " + message);

    if (message == 'off'){
      console.log("test")
      var commande = "docker run \
      --net=host \
      -v /etc/localtime:/etc/localtime:ro \
      -v `pwd`:/usr/src/app \
      tradfri-dev \
      echo 'api(light.light_control.set_dimmer(0))'|python3 -i -m pytradfri 192.168.1.128 \
      ";

      spawn.exec(commande, (err, stdout, stderr) =>{
          if (err){
            console.log(err);
            return;
          }
          console.log(stdout);
      })
    if (message == 'on'){
      var commande = "docker run \
      --net=host \
      -v /etc/localtime:/etc/localtime:ro \
      -v `pwd`:/usr/src/app \
      tradfri-dev \
      echo 'api(light.light_control.set_dimmer(250))'|python3 -i -m pytradfri 192.168.1.128 \
      ";

      spawn.exec(commande, (err, stdout, stderr) =>{
          if (err){
            console.log(err);
            return;
          }
          console.log(stdout);
      })

    }
  });

});

server.listen(PORT, HOST);

console.log('Server listening on '+ HOST +' : '+ PORT);
```

Annexe 5 : Relation serveur et traitement du langage

```java
public class NodeJSEcho {
    // socket object
    public static Socket socket = null;
    public static String message;

    public static void main(String[] args) throws UnknownHostException, IOException, ClassNotFoundException {
        // class instance
        NodeJSEcho client = new NodeJSEcho();

        // socket tcp connection
        String ip = "127.0.0.1";
        int port = 6969;
        client.socketConnect(ip, port);

        // writes and receives the message
        //message = "message123";
        System.out.println("Sending: " + message);
        String returnStr = client.echo(message);
        System.out.println("Receiving: " + returnStr);
    }

    // make the connection with the socket
    public void socketConnect(String ip, int port) throws UnknownHostException, IOException {
        System.out.println("[Connecting to socket...]");
        socket = new Socket(ip, port);
    }

    // writes and receives the full message int the socket (String)
    public String echo(String message) {
        try {
            // out & in
            PrintWriter out = new PrintWriter(getSocket().getOutputStream(), true);
            BufferedReader in = new BufferedReader(new InputStreamReader(getSocket().getInputStream()));

            // writes str in the socket and read
            out.println(message);
            String returnStr = in.readLine();
            return returnStr;
        }
        catch (IOException e) {
            e.printStackTrace();
        }

        return null;
    }

    // get the socket instance
    public Socket getSocket() {
        return socket;
        }
}
```

Annexe 6 : Gestion des commandes de l'utilisateur

```java
package corenlp;


import java.io.IOException;

public class BasicPipelineExample {

    public static BasicPipelineExample test;
    public static String text;
    public static boolean boucle = true;
    public static MyThread t;
    public static int demarrer = 0;

    public static void main(String[] args) throws UnknownHostException, IOException {
        test = new BasicPipelineExample();

        System.out.println("Write your sentence : ");

        while (boucle == true) {
            Scanner sc = new Scanner(System.in);
            String lasttext = getText();
            text = sc.nextLine();
            if (text != lasttext) {
                t = new MyThread(text, test);
                t.start();
                }
        }

    }

    public static String getText() {
        return text;
    }

    public BasicPipelineExample getBasic() {
        return test;
    }
}

```

Annexe 7 : Thread

```
package corenlp;

import java.io.IOException;

public class MyThread extends Thread{

    public static int i;
    public static int k;
    public static int j;
    public BasicPipelineExample test;
    public static String message;
    public static Socket socket;
    public static NodeJSEcho serveur;
    public static List<String> listverb;
    public static List<String> listnom;
    public static boolean boucle = true;
    public static int demarrer = 0;
    public CoreDocument document;
    public String text;
    public static SelectApp app;
    public StanfordCoreNLP pipeline;


    public MyThread(String text,BasicPipelineExample test) {
        i = 0;
        k = 0;
        j = 0;
        this.text = text;
        this.test = test;
        serveur = new NodeJSEcho();
        try {
            serveur.socketConnect("127.0.0.1", 6969);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        pipeline = new StanfordCoreNLP(PropertiesUtils.asProperties("annotators", "tokenize,ssplit,pos", "coref.algorithm","deterministic"));

        // create a document object
        document = new CoreDocument(text);

        // annnotate the document
        pipeline.annotate(document);
    }

      public void run() {

          listverb = new ArrayList<String>();
          listnom = new ArrayList<String>();

          app = new SelectApp(test);


          //text of the first sentence
          String sentenceText = document.sentences().get(0).text();
          System.out.println("Example: sentence");
          System.out.println(sentenceText);
          System.out.println();
          String[] testsentence = sentenceText.split(" ");

          CoreSentence sentence = document.sentences().get(0);

          // list of the part-of-speech tags for the second sentence
          List<String> posTags = sentence.posTags();

          System.out.println("Example: pos tags");
          System.out.println(posTags);
          System.out.println();
```

```
    while (i < testsentence.length) {
        if (posTags.get(i).equals("VBP") || posTags.get(i).equals("VBN") || posTags.get(i).equals("VB") || posTags.get(i).equals("IN")) {

            String verbe = testsentence[i];
            // verbe = "'" + verbe + "'";

            listverb.add(verbe);
            // app.selectAll(verbe);
            i++;
        }
        if (posTags.get(i).equals("NN") || posTags.get(i).equals("NNS") || posTags.get(i).equals("NNP") || posTags.get(i).equals("NNPS")) {

            String nom = testsentence[i];
            // nom = "'" + nom + "'";

            listnom.add(nom);
            // app.selectAll(verbe);
            j++;
            i++;
        } else {
            i++;
        }
    }


    int n = 0;
    int t = 0;
    while (n != listnom.size()) {
        if (listnom.get(n).equals("light") || listnom.get(n).equals("lamp") || listnom.get(n).equals("beacon") || listnom.get(n).equals("illuminant")) {
            while (t != listverb.size()) {
                app.selectAll(listverb.get(t));
                t++;
            }
            n++;
        } else {
            n++;
        }
    }

    message = app.getName();

    serveur.echo(message);


}
```