17/11/2017

Délivrable n°2

Projet ZenBox

Léo Guilpain & François Izabelle & Aguirre Max

Table des matières

Introduction

Le but de ce projet est de créer un boîtier intelligent qui permettra de contrôler des équipements domotiques dans une maison. Ce boîtier devra communiquer avec ces équipements. Nous voulons pouvoir contrôler n'importe quel type d'équipement domotique.

L'interaction entre l'utilisateur et le système se fera à travers une chat box textuelle à l'intérieur d'une application mobile. Cette application devra être accessible par tous les membres d'une famille, utilisant des comptes différents. Un de nos objectifs est de développer cette application sous Android.

Nous voulons une solution domotique intuitive et facile d'utilisation. Elle doit être pratique et on ne veut pas que les données soient stockées sur des serveurs clouds externes à notre système.

Tout d'abord, nous allons faire un point sur l'avancement de notre projet. Dans le premier délivrable, nous avions établi un état de l'art recensant les solutions disponibles sur le marché. Nous avons exposé nos objectifs pour ce projet et avions déjà décomposée notre solution en plusieurs parties distinctes. Nous avions également créé un diagramme de Gantt qui nous permet de suivre l'avancement de notre projet. Certains objectifs ont été atteints, et d'autres non.

	Pla...	Nom de tâche	Durée	Début	Fin
1		**Projet Boîtier Connecté**	**45 jours**	**03/10/2017**	**04/12/2017**
2		Soutenance n°1	6 jours	01/12/2017	08/12/2017
3		Journée Porte Ouverte	6 jours	26/01/2018	03/02/2018
4		**Partie Recherche et Formation**	**34 jours**	**03/10/2017**	**17/11/2017**
5		État de l'art	9 jours	03/10/2017	13/10/2017
6		Recherches sur les différents protocoles de transmission	9 jours	03/10/2017	13/10/2017
7		Formation sur les fonctionnalités de base du Raspberry pi	9 jours	03/10/2017	13/10/2017
8		Détermination du cahier des charges	9 jours	03/10/2017	13/10/2017
9		Affectation des tâches	9 jours	03/10/2017	13/10/2017
10		Détermination des équipements nécessaires	26 jours	13/10/2017	17/11/2017
11		**Partie Communication**	**161 jours**	**13/10/2017**	**25/05/2018**
12		**Délivrable n°1**	**9 jours**	**03/10/2017**	**13/10/2017**
13		Rassembler les informations	9 jours	03/10/2017	13/10/2017
14		Mise en page + webographie / bibliographie	9 jours	03/10/2017	13/10/2017
15		Délivrable n°2	26 jours	13/10/2017	17/11/2017
16		Délivrable n°3	16 jours	17/11/2017	08/12/2017
17		**Mise en ligne d'un site**	**136 jours**	**17/11/2017**	**25/05/2018**
18		Création du site	136 jours	17/11/2017	25/05/2018
19		Skin du site	136 jours	17/11/2017	25/05/2018
20		Mise à jour de l'avancement du projet	136 jours	17/11/2017	25/05/2018
21		**Partie boîtier + Chat Box**	**156 jours**	**20/10/2017**	**25/05/2018**
22		**Développement d'un système basique**	**46 jours**	**20/10/2017**	**22/12/2017**
23		Interprétation de commandes rudimentaires par la chat box	46 jours	20/10/2017	22/12/2017
24		Intégration d'un objet connecté avec le Raspberry pi	46 jours	20/10/2017	22/12/2017
25		**Évolution vers un système plus complexe**	**55 jours**	**08/01/2018**	**23/03/2018**
26		Interprétation de commande plus complexes (Base de données par exemple)	55 jours	08/01/2018	23/03/2018
27		Établir la connexion entre Raspberry pi et tous types d'objets connectés	55 jours	08/01/2018	23/03/2018
28		**Optimisation du système**	**46 jours**	**23/03/2018**	**25/05/2018**
29		Prise en compte d'erreurs dans les demandes de l'utilisateur	46 jours	23/03/2018	25/05/2018
30		Réalisation de fonctions plus complexes	46 jours	23/03/2018	25/05/2018
31		Création du boîtier	100 jours	08/01/2018	25/05/2018

Figure 1 : Ancien Gantt

Nous n'avions ni le recul nécessaire ni une idée assez claire de notre projet lorsque nous avons créé ce Gantt. Après plusieurs semaines d'avancement de notre projet, il est apparu que nous avons sous-estimé la part de préparation et de mise au point. Il nous a fallu

encore quelques semaines afin de décider quels protocoles, librairies et architecture nous allions utiliser. De ce fait, nous avons pris beaucoup de retard par rapport au planning initial. De plus, la méthode de management de l'équipe n'était pas adaptée. Après avoir pris le temps de remettre à plat notre projet et de nous recentrer sur son avancement, nous avons pu dégager une vision plus claire tout en avançant de manière bien plus efficiente qu'auparavant.

De cette phase de prise de recul a émergé un nouveau planning organisationnel sous forme de diagramme de Gantt. Nous avons pris en compte nos erreurs, et nous avons également gagné en compétence ce qui nous a permis de proposer des objectifs plus réalisable et clairs.

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

Figure 2 : Gantt actuel

Dans ce Gantt, on peut voir que dans la partie Tchat Box + Boîtier, 3 grandes sous parties ont été rajoutées : traitement du langage, serveur et client. Nous allons nous focaliser vraiment sur ces parties.

Dans la partie traitement du langage, il va falloir analyser les demandes et être capable de renvoyer les demandes sous le bon format. Ensuite, il faudra entrainer le logiciel.

Dans la partie serveur, il faudra gérer l'authentification et transférer les messages reçus.

Enfin, dans la partie client il faudra dans un premier temps développer une solution test permettant de tester le serveur. Ensuite on développera une application.

Dans ce deuxième délivrable, nous allons vous présenter l'architecture logiciel de notre projet. Nous allons décrire les différentes parties qui vont permettre le bon fonctionnement de notre boîtier. Dans un premier temps, nous allons vous présenter le schéma global de notre architecture logicielle permettant l'explication des différentes relations entre les parties de notre solution. Ensuite nous développerons chaque partie du schéma pour bien en comprendre le fonctionnement.

I. Schéma global

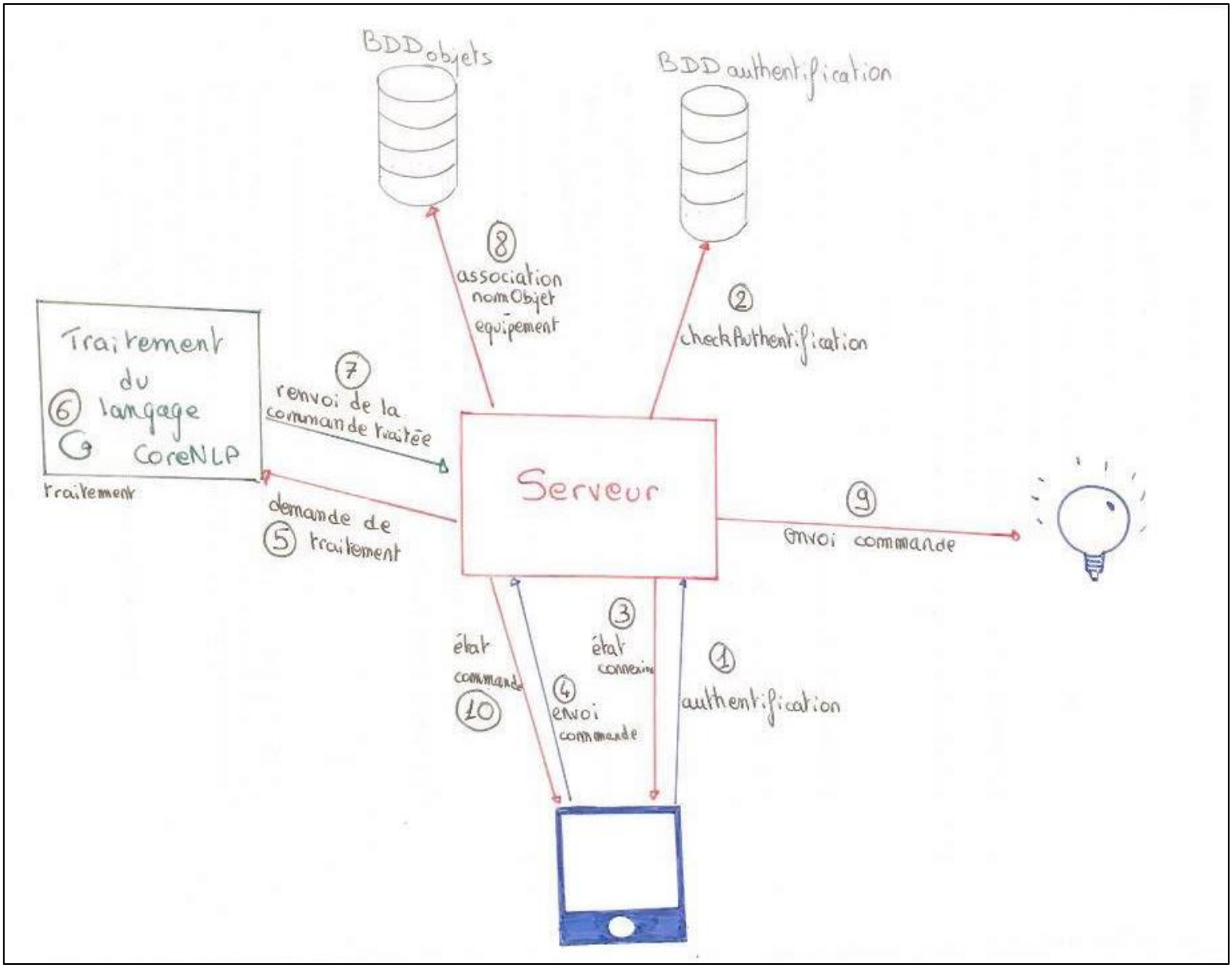

Figure 3 : Schéma global de notre solution

Comme vous pouvez le voir sur ce schéma, dans notre projet, il y a 3 grandes parties :

- La partie Client contenant l'interface utilisateur sous forme d'une application
- Les Objets Connectés de nature variée et communiquant à l'aide de protocoles différents
- Notre Box contenant la partie de traitement du langage et la partie serveur

Dans notre box, on pourra trouver des Raspberry Pi qui feront office de serveur et hébergeront le logiciel de traitement de langage.

Pour commencer (1), le client s'authentifie (pseudo et mot de passe) depuis l'application mobile, ces données seront envoyées à la partie serveur qui les traitera et renverra une réponse permettant la connexion de l'utilisateur et son accès aux fonctions de la ZenBox. Nous utiliserons une base de données afin de stocker les différents pseudos et mot de passe des utilisateurs.

Les commandes seront ensuite envoyées depuis l'application mobile sous forme d'échange de messages textuels avec le serveur. Elles seront d'abord envoyées à la partie serveur qui interagira avec la partie traitement de langage à travers des sockets TCP en utilisant le format JSon pour les messages. Une fois les messages analysés par la partie traitement du langage, ils sont renvoyés sous forme de messages compréhensibles par les la partie serveur qui va pouvoir faire associer les noms d'objets avec les équipements connus par cette partie, afin de les renvoyer aux objets connectés. La partie serveur aura pour rôle d'envoyer les commandes aux différents objets connectés et de renvoyer un message de validation ou d'erreur de la commande à l'utilisateur.

Dans la suite du délivrable, nous allons détailler les différentes parties par leur ordre de priorité de réalisation.

II. Traitement de langage

La partie traitement du langage de notre solution va permettre de comprendre les demandes de l'utilisateur afin de les transformer en commandes compréhensibles par la partie serveur. Entre autres, elle devra comprendre le langage naturel français. Plus précisément, elle doit pouvoir prendre en entrée une chaîne de caractères provenant d'un client et la décomposer afin d'extraire automatiquement les informations importantes avant de les transmettre au serveur sous un format particulier.

Développer une solution de traitement de langage par nous-mêmes est trop compliqué et nous avons choisi d'utiliser un logiciel disponible sur le marché dans le but de traiter cette partie à notre place. Parmi tous ceux que nous vous avions présenté dans le premier délivrable, nous en avons choisi un et c'est celui-là que nous allons vous présenter.

1. Choix du logiciel

Nous avons fait le choix d'utiliser le réseau de neurones créé par l'université de Stanford, Core NLP **[3]**. Nous utiliserons l'API fournie avec CoreNLP. Ce choix repose sur plusieurs facteurs, le premier étant qu'il répond à nos besoins d'un logiciel open source permettant de traiter des données textuelles en français et en Anglais. Ensuite en navigant sur la page web dédié à Core NLP nous avons pu constater qu'il était très bien documenté ce qui facilite la prise en main. Enfin ce qui a motivé notre choix est que tout le code source est en JAVA, langage dans lequel nous avons des connaissances et nous pouvons donc si besoin nous intéresser au code source directement afin de comprendre son fonctionnement. De plus il est possible d'utiliser des wrappers en différents langages ce qui nous permettra d'utiliser celui avec lequel nous somme le plus à l'aise, et celui dans lequel sera développé notre solution. Notre choix se porte sur le langage JavaScript car c'est celui que nous utilisons le plus cette année.
De plus nous avons pu déjà voir en réalisant notre état de l'art qu'il existe déjà des projets ressemblant au notre utilisant CoreNLP sur un Raspberry pi.

2. Fonctionnement de CoreNLP

Le principe du logiciel est de pouvoir utiliser plusieurs outils d'analyse linguistique. Il est possible de choisir lesquels on souhaite utiliser et ceux qui ne correspondent pas à notre demande. Dans ces outils certains permettent au logiciel d'étiqueter les mots selon leurs types (nom, verbe, personne, etc) **[4]**. Enfin une fois les types associés on peut traiter ces données et faire ressortir celles qui nous intéressent. On voit donc bien ici que les fonctionnalités de ce logiciel correspondent pleinement à nos besoins, ce que nous avons pu vérifier grâce à leur version démo sur la page web (voir figure ci-dessous). Pour ce qui est de l'utilisation, il y a la possibilité de tester les différentes fonctions sur des textes directement sur leur site. Une fois que l'on aura pu tester les outils utiles pour notre projet, nous pourrons faire appel à l'outil depuis notre programme. Pour ce faire il y a deux solutions possibles, soit il faut télécharger la librairie correspondant à JavaScript (node.js) depuis le site web, soit utiliser des gestionnaires de dépendances tels que Maven, Ant ou Gradle qui importent automatiquement les librairies.

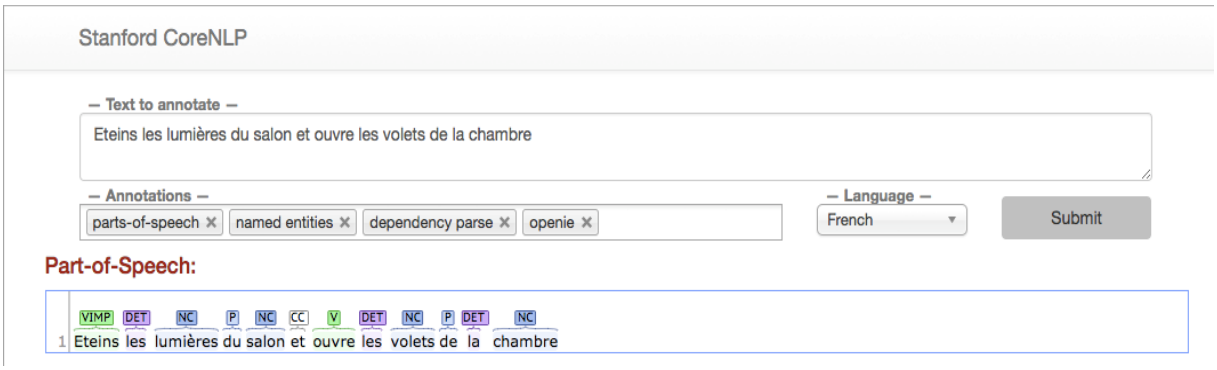

Figure 4 : Test de la reconnaissance d'une commande type sur CoreNLP ***[1]***

3. Utilisation dans notre projet

À partir du schéma global on peut voir que la partie traitement du langage de notre projet reçoit les messages envoyés par le client à travers la partie serveur. Son objectif va être de comprendre les informations fournies dans la commande en repérant les verbes d'actions (exemple : Allumer, Éteindre…) mais aussi l'objet ou le groupe d'objet concerné. Une fois ces informations récupérées la partie traitement du langage va devoir les transformer en une commande au format JSON qui sera envoyé au serveur. Le format JSON est utilisé afin que le serveur qui sera codé en nodeJS puisse l'interpréter facilement. Ce type de document JSon est le suivant :

{pseudo : "nomClient", objet: "idObjet", action : "actionaEffectuer"}

Dans le cas où le scénario est activé par un client, la partie traitement du langage décompose le scénario en plusieurs messages à envoyer à la partie serveur. C'est la partie serveur qui stockera les différents scénarios dans une base de données. Elle enverra les commandes à la partie serveur, et enverra une seule commande pour un seul objet. C'est à dire que lors de l'activation d'un scénario, la partie traitement de langage enverra plusieurs messages à la partie serveur et chacun de ces messages activera un objet connecté.

Cette partie est donc une des plus importantes de notre projet car le bon fonctionnement repose en grande partie sur la compréhension des commandes. Ce sera donc la partie que nous allons devoir réaliser en premier afin de pouvoir la tester au maximum et s'assurer de son bon fonctionnement.

III. Serveur

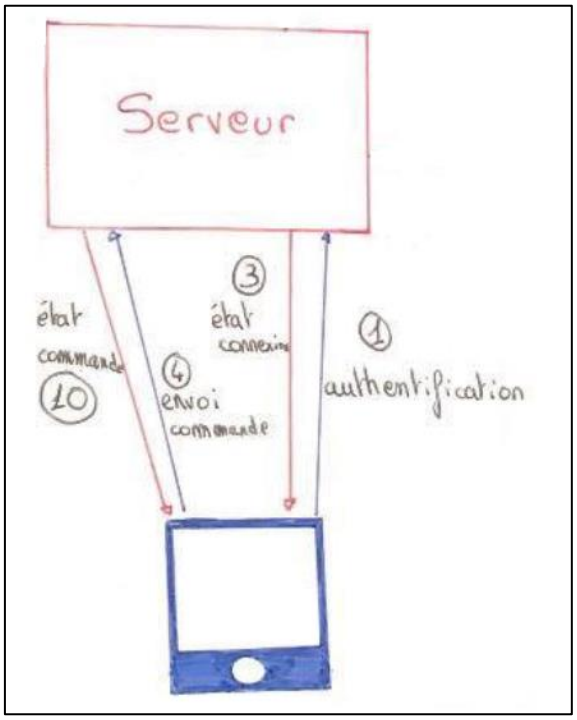

Figure 5 : Schéma des communications avec le serveur

Nous avons choisi de coder notre serveur en JavaScript. Ce choix est basé sur le fait que nous avons vu ce langage en cours et c'est donc le langage que l'on maîtrise le mieux. De plus, dans le cours de WebObject, nous avons fait un serveur de tchat. Nous allons donc utiliser les bases de ce serveur pour concevoir le nôtre. Cela facilitera sa conception. Nous avons également vu que node.JS était compatible avec CoreNLP. Il existe des packages (corenlp) et des wrapper (stanford-corenlp) disponibles. Ainsi, notre partie traitement de langage va pouvoir communiquer sans problème avec le serveur. De plus, vous verrez ensuite que le serveur traite également avec le client. Il existe des librairies compatibles avec Android, cela va donc faciliter la mise en place de la communication.

Dans le serveur, il y aura deux parties, une partie traitement des commandes et une partie base de données. Comme vous l'avez vu, le client va devoir s'authentifier avec un nom d'utilisateur et un mot de passe. Il va falloir stocker ces données. Pour cela, les bases de données relationnelles sont les plus adaptés à notre projet. En effet, pour notre projet, nous aurons juste besoin de stocker le pseudo et le mot de passe de l'utilisateur, donc notre schéma n'aura pas besoin d'être modifié.

Nous allons devoir également stocker les informations concernant les commandes clients et les objets connectés. Nous devrons également stocker les différents scénarios déclenchables

par l'utilisateur. Au final les schémas resteront les mêmes et donc la base de données relationnelle est tout à fait adaptée.

Nous avons choisi d'utiliser sqlite3 car nous l'avons déjà utilisé l'année dernière et donc nous avons des connaissances sur cette base.

Notre serveur servira de broker pour les équipements communiquant en MQTT (voir partie V.B-module).

Notre serveur aura trois rôles dans l'envoi de données :

- Envoyer des données au client
- Envoyer des données aux objets connectés
- Transmettre les commandes clients à la partie traitement du langage

1. Envoyer des données au client

Il y aura deux moments où le serveur enverra des données au client. Le premier est lorsque que le client s'authentifie. Ainsi, lorsqu'il aura reçu les données du client, le serveur stockera ces données dans une base de données puis renverra au client la validité ou non de son authentification.

Dans un second temps, lorsque le client enverra des commandes via l'application mobile afin d'effectuer des actions sur les objets connectés, le serveur renverra l'état de la commande si un retour d'état est disponible avec l'objet en question. C'est-à-dire que le client aura la possibilité de savoir si sa commande a bien été effectuée ou non. On utilise le protocole de communication TCP/ IP.

Ces différents retours d'état seront envoyés sous le format JSON.

Le format JSON d'un message envoyé au client lors d'une authentification sera le suivant :

{nom : « nomClient », action : « authentification », réponse : « acceptée ou non »}

2. Envoyer des données aux objets connectés

Pour les objets connectés, il y aura deux cas de figures. C'est-à-dire qu'il peut y avoir des objets connectés, tels que la lampe ikea tradfri, qui possèdent une box ou un module préinstallé permettant de faire office de relai entre le protocole WIFI et le protocole compris par l'objet connecté. Il y a également d'autres objets qui n'ont pas ce genre de modules. Ainsi le serveur devra être capable de s'adapter. Lorsqu'une box sera présente, le serveur se servira des librairies préalablement installées permettant de communiquer avec la box (ex : libcoap, voir en annexe). Ensuite la box se chargera de communiquer avec l'équipement.

S'il n'y a pas de box, il faudra alors installer un module hardware faisant office de relai permettant au serveur de communiquer avec les différents équipements.

3. Recevoir des données du client

Lorsque le client veut envoyer une commande elle passe d'abord par notre partie serveur avant d'être transférée à la partie traitement du langage. Le serveur recevra donc les commandes client en brut.

4. Recevoir/Envoyer les données de la partie traitement du langage

Une fois que les données seront traitées, les messages seront renvoyés par la partie traitement langage vers la partie serveur. Le serveur ira comparer les données reçues à celles de sa base de données afin d'envoyer les commandes aux objets connectés adéquats.

Les messages seront également transférés à la partie traitement du langage en brut.

5. Test du serveur

Afin de pouvoir tester le bon fonctionnement de notre serveur nous utiliserons un programme Javascript exécutable avec node.js. Elle pourra communiquer avec la partie serveur afin d'effectuer des commandes clients et pour l'authentification. Cette partie sera donc la partie client de notre solution. Les messages se transmettront au travers de sockets TCP et seront au format JSON. Les fonctions implémentées seront les suivantes :

1. Le client test doit pouvoir demander la liste des objets connectés et leur état s'ils peuvent le fournir.
2. Il est possible de modifier l'état d'un objet (allumer ou éteindre une lampe, etc.). Il n'est possible de modifier que l'état d'un équipement à la fois. La possibilité de créer des scènes ne sera implémentée que plus tard.
3. Le client doit pouvoir quitter l'application proprement.
4. Le client doit pouvoir parler en langage naturel. Il doit être restreint le moins possible sur le format des demandes.

Le format JSON d'un message envoyé à la partie traitement du langage lors d'une commande client sera le suivant :

{nom : « nomClient », action : « commande du client », objet : « nomObjet »}

La commande du client transmise sera non modifiée. C'est la partie traitement du langage qui s'en chargera.

Le format JSON d'un message dans le cadre de l'authentification sera le suivant :

{action : « connection », pseudo : « pseudoINPUT », mdp : « mdpINPUT »}

Toutes les données échangées seront cryptées, selon le modèle exposé dans la partie sécurité.

IV. Client

Dans un premier temps notre partie client sera très basique comme nous l'avons déjà expliqué dans la partie serveur et servira à tester le bon fonctionnement des parties serveur et traitement du langage. Autrement dit, le traitement d'une demande client étant la partie essentielle dans notre projet ; il faut qu'elle puisse être testée le plus rapidement possible. Cependant une fois que les parties serveurs et traitement du langages seront opérationnelles nous souhaitons réaliser une Application intuitive pour rendre son utilisation facile pour l'utilisateur.

2.1. Plateforme utilisée

La solution finale de notre application sera une application mobile développée sous plateforme Android. Suivant des cours de développement d'application mobile sous Android,

nous avons ou aurons les compétences suivantes nécessaires afin d'atteindre nos objectifs d'ici la fin du semestre :

- Connaissance des langage Java et Javascript
- Connaissance de l'environnement de travail Android Studio
- Utilisation du protocole Bluetooth

Nous pensons donc avoir les compétences nécessaires pour atteindre nos objectifs en utilisant cette solution. Cependant, un de nos objectifs étant de rendre possible l'utilisation de notre solution sur d'autres plateformes qu'Android, nous devrons plus tard nous former afin d'acquérir les compétences nécessaires.

Concernant le choix du protocole de communication utilisé entre notre partie client et les autres parties de notre solution, nous avions le choix entre le Bluetooth et le WIFI. En effet ces deux protocoles étaient tout à fait adaptés à notre solution et pouvaient facilement être utilisés avec des cartes Raspberry Pi. N'ayant pas d'intérêt à choisir l'un plutôt que l'autre, nous avons choisi Bluetooth sans raison particulière

2.2. Schéma de notre solution

Vous trouverez en **annexe 2** notre schéma global de l'application.

Ce schéma ne représente que les différentes interfaces graphiques auxquelles sera confronté l'utilisateur final. Il ne représente pas l'échange de données entre notre application mobile et la partie traitement de langage ou serveur. L'échange de messages se fait de la même manière que celle employée dans la solution test. Cette architecture nous convient puisqu'elle permet de couvrir tous besoins. Nous pourrons l'améliorer par la suite si besoin sans problèmes puisque les différentes parties sont bien séparées.

Lorsque le client se connecte à l'application, il arrive sur un écran où il peut renseigner son Login et Mot de passe afin de s'authentifier. Ensuite, il pourra choisir de configurer des scènes ou alors d'accéder au chat qui lui permettra de communiquer avec notre solution domotique. Dans la partie éditeur, il sera possible de renseigner des mots-clefs particuliers qui déclencheront les scènes afin de faciliter le travail de la partie de traitement du langage. Par exemple on pourra spécifier que le mot clef "salon" référera à tous les objets connectés situés dans le salon (qui ne seront connus que par un identifiant et non par leur position géographique). Ces mots clés seront stockés dans la partie serveur qui s'occupera de séparer une commande de scènes en plusieurs commande.

2.3. Communications avec le serveur

a. Authentification

La partie application mobile de notre solution devra envoyer des données à notre partie serveur dans le cadre de l'authentification. La partie serveur gérera la base de données contenant tous les comptes utilisateurs autorisés à se connecter au système.

Le protocole utilisé sera du Bluetooth, ce qui permet de communiquer sans problème avec la partie serveur où que soit l'utilisateur dans la limite de la propriété.

Le format du message sera bien spécifié puisque les messages seront transmis sous un format JSON particulier. Le format exact des messages est le même que celui décrit dans la solution test.

b. Envoie de commande

La partie application mobile devra aussi envoyer des données au serveur dans le cadre d'envoi de commandes par l'utilisateur.
Ces commandes seront envoyées par protocole Bluetooth également.

Le message sera encore « brut », c'est-à-dire que la commande utilisateur ne sera pas encore modifiée. C'est la partie traitement du langage qui s'en chargera. Le message encapsulé dans le protocole de transmission sera tout de même au format JSON, de la même manière qu'expliqué dans la partie solution test.

V. Plugin et module

Dans cette partie nous allons faire la différence entre plugin et module. Un plugin sera défini comme une partie software de notre solution : c'est ce qui permettra de faire la liaison au niveau programme entre notre serveur et les modules utilisés. Ce sont par exemple les librairies que l'on va utiliser afin de communiquer avec les modules. Les modules quant à eux sont une partie hardware de notre solution. Ce sont les équipements qui nous permettront de communiquer avec les objets connectés, comme une carte wifi ou un module Zigbee.

1.Plugins

Nous utiliserons des plugins adaptés aux objets connectés que nous aurons à notre disposition. Les plugins utilisés seront les suivants :

- **Libcoap** : pour les objets IKEA. Cela nous permettra de communiquer avec la passerelle de la marque qui communiquera ensuite en Zigbee avec ses objets connectés. Le protocole utilisé par la passerelle est le protocole CoAP encapsulé dans du HTTP. Libcoap est une librairie permettant de communiquer en CoAP et ainsi de transmettre nos commandes de manières compréhensibles pour la passerelle IKEA. (**voir annexe 1**)
- **Mqtt** : C'est un protocole pratique pour faire communiquer des objets connectés entre eux. Il fonctionne avec des "brokers", c'est à dire que chaque équipement peut s'abonner à un topic et recevoir tous les messages publiés dans ce topic. Le broker est le serveur qui va distribuer les messages aux objets connectés. C'est un protocole léger puisque lorsque l'on veut envoyer un message à plusieurs clients il suffit de poster le message sur le topic, et les paquets envoyés ne sont pas surchargés en en-têtes comme le protocole HTTP. Une librairie MQTT.js est disponible avec node.js. Il nous faudra configurer un intermédiaire mqtt-radio qui permettra de communiquer avec l'équipement, s'il n'embarque pas déjà de client mqtt. **[2]**

2.Modules

Nous aurons besoin de plusieurs modules afin de communiquer avec nos équipements. Certains équipements communiqueront via Zigbee (sans être de la marque IKEA) et donc nous

aurons besoin d'un module Zigbee. Il en est de même pour la carte wifi dont nous aurons besoin pour communiquer avec les objets connectés son-off. Cependant, les Raspberry Pi V3 intègre déjà une carte wifi, nous nous adapterons donc en fonction de la version des cartes Raspberry à notre disposition.

VI. Objets connectés

Les objets connectés que nous aurons à notre disposition seront des marques suivantes :

- Ikea. Ces équipements fonctionnent grâce au protocole Zigbee et communiquent avec une box qui centralise tous les objets connectés de la marque. Cette box communique avec l'application via le protocole CoAP encapsulé dans du HTTP. Nous avons deux possibilités pour contrôler les objets connectés de la marque Ikea. Premièrement, nous pouvons contrôler la box en utilisant Libcoap afin de "court-circuiter" les échanges entre l'application et la box. La seconde possibilité est d'interagir en Zigbee directement avec les objets connectés. Cela nous évite de devoir utiliser la box.
- Sonoff : ces équipements fonctionnent par l'intermédiaire de cartes wifi intégrées. Il est possible de modifier le code des équipements pour qu'ils puissent communiquer avec notre partie serveur en ajoutant par exemple un client mqtt.

VII. Sécurité

1. Importance de la sécurité

La sécurité est une partie importante dans le domaine de l'IOT de nos jours. En effet, il est nécessaire que les communications avec les objets connectés soient protégées car seuls les propriétaires doivent pouvoir accéder à leur contrôle ainsi qu'à leurs données.

Nous allons donc devoir nous intéresser à la sécurité à chaque endroit où ont lieu des communications car ce sont à ces endroits qu'il est possible de récupérer des informations voir d'envoyer de fausses informations.

2. Protection de nos données

Nous allons utiliser nos connaissances issues du module SRIO dans lequel nous avons pu voir les différents types d'attaques possibles et comment il était possible de limiter les risques liés au hacking.

Tout d'abord pour l'accès à notre application il sera nécessaire pour l'utilisateur de s'authentifier afin que toute personne passant près de l'installation ne puisse pas avoir accès au contrôle de ses différents équipements connectés.

Pour ce qui est des communications entre les différents éléments visibles sur notre schéma les données devront être cryptées afin qu'une personne ne puisse pas récupérer toutes les informations sur le fonctionnement de la ZENBOX simplement en écoutant les communications. Nous utiliserons un système de clé asymétrique couplé à un système de signature afin de respecter les contraintes suivantes de la sécurité :

- Confidentialité : Gérée par le cryptage des données
- Intégrité : le système de clefs asymétriques permet de s'assurer que le message ne puisse être modifié par une personne extérieure au système.
- Authentification : Notre système d'authentification s'assurera de cette problématique. Il s'assurera que l'émetteur d'une commande soit bel et bien celui ayant envoyé cette commande.
- Non-répudiation : Le système de signature nous permettra de s'assurer que l'émetteur d'une commande ne puisse pas nier l'avoir envoyée.

Concernant la sécurité des objets connectés qui possèdent une box, nous nous renseignerons sur leur protocole de sécurité afin de savoir si on peut leur faire confiance ou non. Si nous estimons que les objets ne sont pas assez sécurisés, nous ne les utiliserons pas.

Conclusion

À ce stade de notre projet, nous avons une vision très claire de nos objectifs et des différentes parties à mettre en place afin de pouvoir assurer un bon fonctionnement de notre solution.

Nous avons bien défini les différentes briques technologiques ainsi que les protocoles de communication que nous allons utiliser. Pour la plus parts nous les avons déjà testés de manières simples afin de vérifier la cohérence de leur utilisation vis à vis de nos objectifs.

Pour ce qui est de la suite il va falloir pousser plus loin ces tests et commencer à développer les parties essentielles comme la partie traitement du langage et la partie serveur. Ces deux parties sont les composantes de la box et peuvent être testées facilement sans avoir besoin du reste. De cette manière dans un second temps il ne nous restera que les communications et l'application mobile à mettre en place.

Références

[1] «Corenlp.run,» [En ligne]. Available: http://corenlp.run/.

[2] «MQTT,» [En ligne]. Available: http://mqtt.org/.

[3] «Stanford CoreNLP,» [En ligne]. Available: https://stanfordnlp.github.io/CoreNLP.

[4] «Stanford natural language,» [En ligne]. Available: https://www.supinfo.com/articles/single/4726-stanford-natural-language-processing-nlp.

Annexe 1 : Libcoap

***coap-client* [*-A* type1, _type2_ ,...] [*-t* type] [*-b* [num,]size]**
[*-B* seconds] [*-e* text] [*-f* file] [*-m* method] [*-N*]
[*-o* file] [*-P* addr[:port]] [*-p* port] [*-s* duration]
[*-O* num,text] [*-T* token] [*-v* num] [*-a* addr] [*-U*] URI

***-a* addr::**
The local address of the interface that has to be used.

***-b* [num,]size::**
The block size to be used in GET/PUT/POST requests (value must be a multiple of 16 not larger than 1024 as libcoap uses a fixed maximum PDU size of 1400 bytes). If 'num' is present, the request chain will start at block 'num'. When the server includes a Block2 option in its response to a GET request, coap-client will automatically retrieve the subsequent block from the server until there are no more outstanding blocks for the requested content.

***-e* text::**
Include text as payload (use percent-encoding for non-ASCII characters).

***-f* file::**
File to send with PUT/POST (use '-' for STDIN).

***-m* method::**
The request method for action (get|put|post|delete), default is 'get'.

***-o* file::**
A filename to store data retrieved with GET.

***-p* port::**
The port to listen on.

***-s* duration::**
Subscribe to the resource specified by URI for the given 'duration' in seconds.

***-t* type::**
Content format for given resource for PUT/POST. 'type' must be either a numeric value reflecting a valid CoAP content format or a string describing a registered format. The following registered content format descriptors are supported, with alternative shortcuts given in parentheses:

text/plain (plain)
application/link-format (link, link-format)

application/xml (xml)
application/octet-stream (binary, octet-stream)
application/exi (exi)
application/json (json)
application/cbor (cbor)

***-v* num::**
The verbosity level to use (default: 3, maximum is 9).

-A type::
Accepted media types as comma-separated list of symbolic or numeric values, there are multiple arguments as comma separated list possible. 'type' must be either a numeric value reflecting a valid CoAP content format or a string that specifies a registered format as described for option *-t*.

***-B* seconds::**
Break operation after waiting given seconds (default is 90).

***-N* ::**
Send NON-confirmable message. If option *-N* is not specified, a confirmable message will be sent.

***-O* num,text::**
Add option 'num' with contents of 'text' to the request.

***-P* addr[:port]::**
Address (and port) for proxy to use (automatically adds Proxy-Uri option to request).

***-T* token::**
Include the 'token' to the request.

***-U* ::**
Never include Uri-Host or Uri-Port options.

Annexe 2 : Schéma des possibilités de l'interface utilisateur de l'application client