Objets connectés: stratégies et technologies pour une interaction réussie

Théorie, expérimentations et implémentation d'un scénario avec Spotify

Travail de Bachelor

Julien Clément
Août 2018

Supervisé par :

Prof. Dr. Jacques Pasquier-Rocha
and
Arnaud Durand
Software Engineering Group

UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Groupe Génie Logiciel
Département d'Informatique
Université de Fribourg (Suisse)

Software Engineering Group

Remerciements

Je tiens à remercier sincèrement le Software Engineering Group de l'Université de Fribourg qui m'a encadré et soutenu dans la réalisation de ce travail.

J'adresse un merci particulier à Arnaud Durand pour son temps, sa disponibilité et ses excellents conseils qui m'ont été d'une grande aide.

Résumé

Dans un monde où la technologie évolue sans cesse, de nouveaux acteurs ont fait leur apparition et se démocratisent depuis peu : les objets connectés. L'Internet of Things et le Web of Things deviennent dès lors des notions centrales que la plupart des acteurs du marché souhaitent développer. Dans cette quête, les fabricants produisent des standards logiciels presque tous différents les uns des autres. A l'heure actuelle, combiner plusieurs appareils ne se fait pas sans mal et requiert un travail en amont.

Après une introduction aux technologies permettant aux objets connectés de communiquer, plusieurs implémentations possibles se servant de matériel différent seront étudiées afin de cibler les points positifs et négatifs de chacune. Cette mise en pratique permettra de définir la meilleure solution du point de vue d'un développeur.

Finalement, ces recherches seront mises à profit dans un cas concret. Un service web mêlant plusieurs objets connectés et le service de streaming musical Spotify sera implémenté.

Mots-clés : Web of Things, Internet of Things, Smart Things, Bluetooth Low Energy, GATT, MQTT, Thingy, Raspberry Pi, ESP32, REST, Websockets, Node.js, JavaScript

Table des matières

1. Introduction **2**

1.1. Contexte, motivation et objectifs 2
1.2. Organisation 2
1.3. Notations et conventions 3

2. Technologies actuelles et détails techniques **4**

2.1. Les grandes lignes 4
2.1.1. Objets connectés ou intelligents 4
2.1.2. IoT et WoT 5
2.1.3. Problématique 5
2.2. Au coeur des objets connectés 5
2.2.1. BLE : Bluetooth Low Energy 5
2.2.2. GATT : Generic Attribute Profile 5
2.3. Interagir avec les objets connectés 7
2.3.1. MQTT : Message Queuing Telemetry Transport 7
2.3.2. Bluetooth direct 9

3. Expérimentations **10**

3.1. Matériel utilisé 11
3.1.1. Objets connectés 11
3.1.2. Points d'accès 12
3.1.3. Langages utilisés 12
3.2. Premiers pas : noble-device 13
3.2.1. Architecture 13
3.2.2. Fonctionnalités 13
3.2.3. Limitations 13
3.3. Raspberry Pi : EspruinoHub, MQTT 14
3.3.1. Architecture 14
3.3.2. Fonctionnement 14
3.3.3. Limitations 14

3.4. Raspberry Pi : Node-RED, EspruinoHub, MQTT 15
3.4.1. Architecture 15
3.4.2. Fonctionnement 15
3.4.3. Limitations 16
3.5. ESP32 : esp32-ble2mqtt, MQTT 16
3.5.1. Architecture 16
3.5.2. Fonctionnement 17
3.5.3. Limitations 17
3.6. ESP32 : esp32-ble2mqtt, Mozilla Gateway 17
3.6.1. L'implication de Mozilla dans l'IoT 17
3.6.2. Intégration du Thingy :52 au Mozilla Gateway 19
3.6.3. Limitations 20
3.7. Web Bluetooth API 21
3.7.1. Architecture 21
3.7.2. Fonctionnement 21
3.7.3. Implémentation 21
3.7.4. Limitations 22
3.8. Conclusion 22

4. Application des expérimentations : Objets BLE et Spotify 24
4.1. Motivation 24
4.2. Concept 24
4.2.1. Connexion via Spotify : Fonctionnalités 25
4.2.2. Connexion à une session en cours : Fonctionnalités 28
4.3. Spotify Web API 28
4.3.1. Connexion et vie privée 28
4.3.2. spotify-web-api-js 29
4.4. Implémentation 29
4.4.1. Code source 29
4.4.2. Lecteur musical 29
4.4.3. Mood 30
4.4.4. Party 30
4.5. Extensions et améliorations 31
4.6. Démonstration 31

5. Conclusion 32
5.1. Synthèse 32
5.2. Evolution future 32

A. Acronymes communs 34

Références 35

Liste des figures

2.1. Service GATT prédéfini : Heart rate service 6
2.2. Read characteristic - nRF Connect 6
2.3. Write characteristic - nRF Connect 7
2.4. Notify characteristic - nRF Connect 7
2.5. MQTT 8

3.1. Nordic Thingy :52 11
3.2. Ampoule LED Magic Blue 11
3.3. iTAG 11
3.4. Raspberry Pi 3 Model B 12
3.5. ESP32 12
3.6. Architecture - Noble-device 13
3.7. Fonctionnalités - SwaggerUI 13
3.8. Valeurs du capteur de température d'un Thingy. JSON. 13
3.9. Architecture - Raspberry Pi - EspruinoHub 14
3.10. Architecture - Raspberry Pi - Node-RED 15
3.11. Node-RED : exemple de flux 16
3.12. Architecture - ESP32 : esp32-ble2mqtt 16
3.13. ESP32 : esp32-ble2mqtt - Topics MQTT et valeurs 17
3.14. Mozilla IoT 19
3.15. Architecture - Mozilla Gateway 19
3.16. Mozilla Gateway - Thingy :52 20
3.17. Architecture - Web Bluetooth API 21
3.18. My Dashboard - Tableau de bord basé sur l'API Web Bluetooth 23

4.1. Interface - Connexion 25
4.2. Interface - Partie Spotify 25
4.3. Interface - Partie Bluetooth 26
4.4. Mode Mood - Transitions 27
4.5. Interface - Mode Party 28

Liste des tableaux

2.1. Topics MQTT 9

Liste des codes source

3.1. WoT Thing Description. JSON. [15] 18
3.2. Web Thing API. JSON. [18] 18
3.3. Connexion et souscription aux notifications d'une caractéristique d'un Thingy. JavaScript. 21
3.4. Changement de la couleur d'une ampoule Magic Blue en rouge. JavaScript. 22

4.1. Ajout d'une chanson aux chansons sauvegardées. JavaScript. 29
4.2. Gestion du bouton du Thingy. JavaScript. 30
4.3. Gestion d'un vote positif côté serveur. Node.js. 31

1
Introduction

1.1. Contexte, motivation et objectifs 2
1.2. Organisation 2
1.3. Notations et conventions 3

1.1. Contexte, motivation et objectifs

L'évolution de la technologie révolutionne notre quotidien depuis des décennies. Certains objets dont l'utilité réelle fut questionnée lors de leur apparition font désormais partie intégrante de nos vies et redéfinissent nos façons de travailler, communiquer et nous divertir : l'objectif principal étant de simplifier la vie de tout un chacun.

Ces changements font apparaître une nouvelle catégorie de produits : les objets connectés. Grâce à eux, l'interaction entre nos systèmes informatiques et le monde réel devient possible. Qu'ils se trouvent dans nos maisons, nos voitures ou encore accrochés à nos poignets, ils semblent offrir une infinité de façons de les intégrer à nos vies, uniquement limitées par notre imagination. Qu'en est-il en pratique ? Quelles sont les technologies actuelles et comment les utiliser ? Sont-elles accessibles et fiables ? Est-il aisé de les associer ?

L'objectif de ce travail visera dès lors à étudier, comparer et combiner les technologies existantes afin d'observer les éléments à prendre en compte et les éventuelles difficultés découlant du développement logiciel dans le cadre de l'Internet of Things, avant d'implémenter un scénario concret faisant interagir divers objets connectés avec le service de streaming musical Spotify.

1.2. Organisation

Introduction

L'introduction contient les motivations et objectifs de ce travail, ainsi qu'une vue d'ensemble des thèmes traités.

Chapitre 1 : Technologies actuelles et détails techniques

Ce chapitre part d'une vue globale du monde des objets connectés en définissant des concepts centraux comme l'IoT et le WoT pour progressivement explorer leur fonctionnement interne et leurs modes de communication.

Chapitre 2 : Expérimentations

Cette deuxième partie étudie différentes technologies et manières d'interagir avec plusieurs objets connectés. Leur fonctionnement, leurs forces et leurs faiblesses seront exposés afin d'identifier la meilleure architecture et implémentation selon certains critères.

Chapitre 3 : Objets BLE et Spotify

Les connaissances acquises grâce aux expérimentations des chapitres précédents sont mises en pratique dans l'implémentation d'un service web où les objets connectés deviennent un simple outil et interagissent avec le service de streaming musical Spotify.

Chapitre 4 : Conclusion

La conclusion synthétise le travail effectué et répond aux interrogations émises précédemment. Elle évoque des points récurrents liés au monde des objets connectés et donne des pistes quant à l'évolution future du domaine.

Appendix

Contient une liste des acronymes employés ainsi que les références sur lesquelles s'appuie ce travail.

1.3. Notations et conventions

— Conventions de format :

  — Les abbréviations et acronymes sont sous la forme Hypertext Transfer Protocol (HTTP) pour le premier usage et HTTP pour les suivants ;

  — `http://localhost:8888` est utilisé pour les adresses web ;

  — Le code est formaté de la façon suivante :

```
1 public double division(int _x, int _y) {
2   double result;
3   result = _x / _y;
4   return result;
5 }
```

— Le travail est séparé en 5 chapitres, eux-mêmes subdivisés en sections, sous-sections et sous-sous-sections. Ces dernières sont organisées en paragraphes, signifiant des coupures logiques.

— Les objets de type Figure, Table et Listing sont numérotés au sein des chapitres. Par exemple, une référence à Figure j du chapitre i aura la notation *Figure i.j*.

— En ce qui concerne le genre, la forme masculine est constamment employée par simplicité. Les deux genres sont considérés de façon égale.

2

Technologies actuelles et détails techniques

2.1. Les grandes lignes 4
2.1.1. Objets connectés ou intelligents 4
2.1.2. IoT et WoT 5
2.1.3. Problématique 5
2.2. Au coeur des objets connectés 5
2.2.1. BLE : Bluetooth Low Energy 5
2.2.2. GATT : Generic Attribute Profile 5
2.3. Interagir avec les objets connectés 7
2.3.1. MQTT : Message Queuing Telemetry Transport 7
2.3.2. Bluetooth direct 9

2.1. Les grandes lignes

2.1.1. Objets connectés ou intelligents

Les objets connectés, également appelés objets intelligents, sont des objets physiques possédant une ou plusieurs des spécificités suivantes : des capteurs (température, mouvement, etc.), des actuateurs (son, écran, etc.), une capacité de calcul, des interfaces de communication. [1]

Nombreux sont les exemples faisant désormais partie du quotidien de la personne lambda : wearables (montres connectées, traqueurs d'activité, etc.), assistants vocaux (Google Home, Amazon Echo, etc.), éclairage intelligent (Philips Hue, etc.), détecteurs de mouvement, caméras de surveillance, serrures intelligentes, etc. Bien que certains objets visent encore un public intéressé, leur facilité apparente d'utilisation tend à les démocratiser.

[1] Guinard, Dominique D. and Trifa, Vlad M., Building the Web of Things [2]

2.1.2. IoT et WoT

L'Internet of Things (IoT) définit un système d'objets physiques pouvant être découverts, surveillés, contrôlés, avec lesquels il est possible d'interagir à l'aide d'appareils électroniques qui communiquent via diverses interfaces réseau, ou qui peuvent éventuellement être connectés à Internet au sens plus large. [2]

Le Web of Things (WoT) représente quant à lui l'association de ces objets avec des technologies web, ce qui permet d'outrepasser les limitations (notamment spatiales) de l'IoT et de communiquer avec eux à distance.

2.1.3. Problématique

De plus en plus d'entreprises de renom se lancent dans la conception d'objets connectés. La conception de matériel informatique nécessite évidemment une partie logicielle pour lui permettre de transmettre les informations relevées vers d'autres appareils. De ce fait, la majorité des fabricants décide de proposer sa solution logicielle, son protocole de communication et son standard, ce qui cause une hétérogénéité entre les produits disponibles sur le marché. Ainsi, l'utilisateur final est actuellement contraint de se limiter à l'écosystème d'un fabricant, ou du moins, de se renseigner outre mesure sur la compatibilité de ses produits. L'idéal serait donc de proposer une solution unique, un standard permettant à quiconque de configurer facilement n'importe quel objet intelligent acheté sur le marché. Il sera donc question de trouver des solutions accessibles pour mettre en relation divers objets connectés.

2.2. Au coeur des objets connectés

2.2.1. BLE : Bluetooth Low Energy

Une mauvaise autonomie est un point rédhibitoire pour le succès d'un objet connecté. Idéalement, il devrait pouvoir être fonctionnel en tout temps sans que l'utilisateur n'ait à se préoccuper de son niveau de batterie. Comme son nom l'indique, le Bluetooth Low Energy (BLE) est une solution de communication sans fil qui a la particularité d'être peu énergivore. Par exemple, l'autonomie d'un beacon peut s'étendre à plusieurs années en s'appuyant sur une simple pile bouton comme source d'énergie. Le BLE est actuellement supporté par la majorité des smartphones et ordinateurs du marché et devient la technologie indispensable en IoT.

2.2.2. GATT : Generic Attribute Profile

Vue d'ensemble

La communication sans fil entre un objet Bluetooth Low Energy et un autre périphérique est en principe définie par le standard Generic Attribute Profile (GATT), construit sur l'Attribute Protocol (ATT). Il est nécessaire de connaître les notions suivantes pour appréhender la communication via BLE :

[2] Guinard, Dominique D. and Trifa, Vlad M., Building the Web of Things [2]

— **Profil :** Un profil est une collection de services.

— **Service :** Un service est un ensemble de caractéristiques qui englobe un comportement particulier de l'appareil. Chaque service possède un identifiant UUID qui peut soit correspondre à un UUID prédéfini dans le standard GATT (16 bits) comme le "Battery Service", soit être défini par le fabricant pour un service "personnalisé" (128 bits).

— **Caractéristique :** Une caractéristique représente un attribut qui possède une certaine valeur basée sur un capteur ou spécifiée par l'utilisateur. Par exemple, un "Environment Service" contiendra une caractéristique "Temperature" ou "Light", dont la valeur dépendra des capteurs de l'appareil, tandis que le "Battery Service" prédéfini contiendra la caractéristique "Battery Level". Chaque caractéristique possède également un UUID particulier qui, par convention, ne diffère que peu de l'UUID du service auquel elle est rattachée.

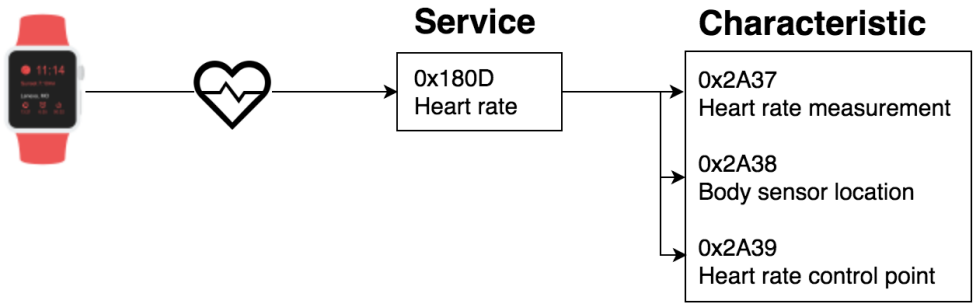


Figure 2.1. – Service GATT prédéfini : Heart rate service

Caractéristiques : propriétés

Les caractéristiques sont divisées selon trois propriétés. Une même caractéristique peut en posséder une ou plusieurs :

— **Read :** En accédant à une caractéristique donnée dans un service donné, la valeur de celle-ci est lue une fois. Pour obtenir une valeur mise à jour, il faut à nouveau lire la caractéristique. Exemple : Nom de l'appareil.

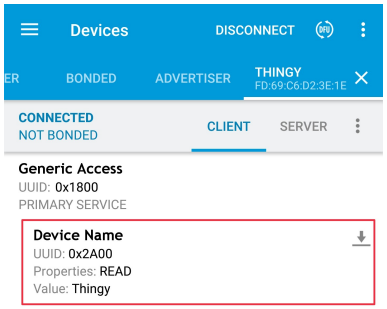


Figure 2.2. – Read characteristic - nRF Connect

— **Write :** L'utilisateur peut modifier la valeur associée à une caractéristique, ce qui influera sur le comportement de l'appareil. Exemple : couleur d'une LED.

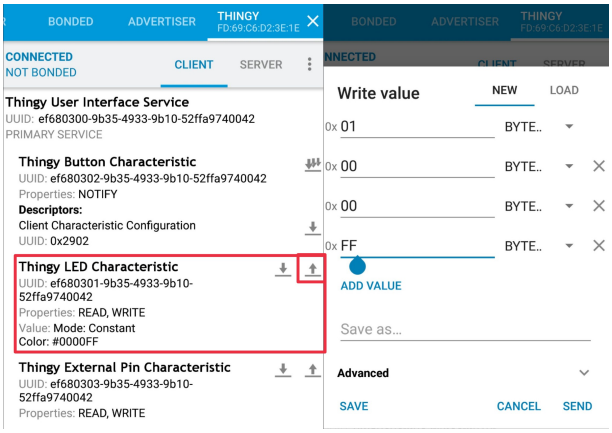


Figure 2.3. – Write characteristic - nRF Connect

— **Notify :** En souscrivant aux notifications d'une caractéristique, l'utilisateur recevra un message contenant la valeur actualisée de cette dernière à chaque fois qu'un changement surviendra. Il est généralement possible de définir la fréquence à laquelle les valeurs sont relevées par l'appareil. Exemple : capteur de température.

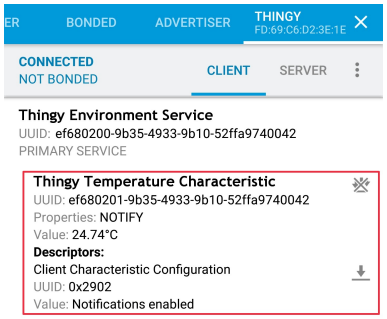


Figure 2.4. – Notify characteristic - nRF Connect

2.3. Interagir avec les objets connectés

Dans le chapitre 3, des exemples concrets des technologies décrites dans cette section seront implémentés afin de communiquer avec les objets connectés.

2.3.1. MQTT : Message Queuing Telemetry Transport

Fonctionnement

Le protocole Message Queuing Telemetry Transport (MQTT), illustré sur la figure 2.5, fonctionne selon un modèle "publish/subscribe" (publication/souscription). Concrètement, un "broker" MQTT joue le rôle de point d'accès central. Tous les clients communiquent avec lui. Pour lire ou écrire une valeur, ils doivent impérativement s'incrire à ou publier sur un topic spécifique.

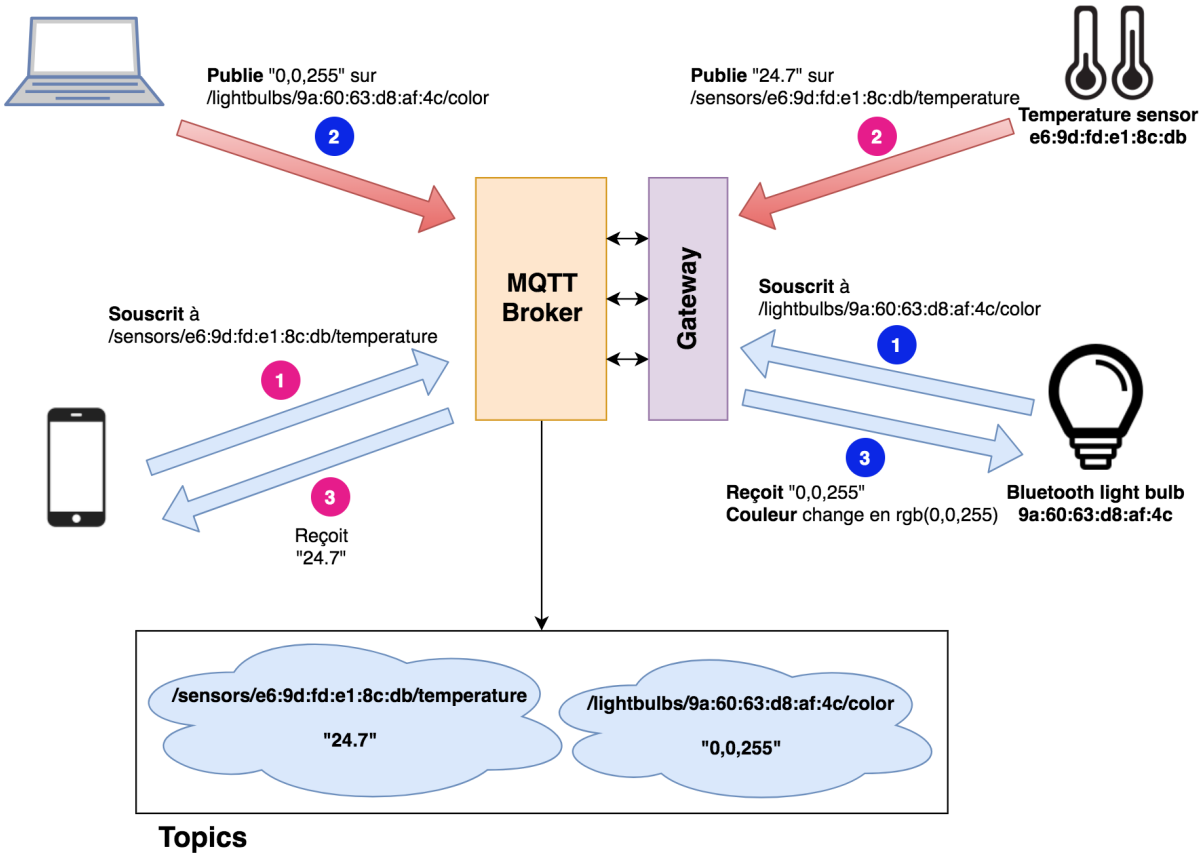


Figure 2.5. – MQTT

Topics : jokers ou wildcards

La publication d'un message se fait impérativement sur un topic précis. A contrario, la souscription à un topic peut inclure de la généralité grâce aux jokers "+" (1 niveau) et "#"(plusieurs niveaux).

Soit le scénario suivant : 2 appareils connectés (EF-00-58-93-F9-9F et 1B-68-78-18-A2-E8) captent la température et la pression de l'air environnant. Ils publient leurs captations sur les topics "/sensors/DEVICE/temperature" et "/sensors/DEVICE/pressure" respectivement, où "DEVICE" est l'adresse MAC de l'appareil. Les effets de la souscription à différents topics sont décrits dans le tableau 2.1.

Topic	Valeurs récupérées
/#	Tous les messages. Ici, température et pression des deux appareils, mais aussi, suivant la configuration du broker, les messages "advertise" des appareils à proximité, etc.
/sensors/+/temperature	Température des deux appareils
/sensors/EF-00-58-93-F9-9F/temperature	Température du premier appareil
/sensors/EF-00-58-93-F9-9F/+	Température et pression du premier appareil
/sensors/EF-00-58-93-F9-9F/#	Température et pression du premier appareil
/sensors/+/+	Température et pression des deux appareils
/sensors/+	Rien
/sensors/#	Température et pression des deux appareils
/sensors/#/temperature	Topic invalide

Table 2.1. – Topics MQTT

Points forts

Premièrement, la notion de Quality of Service (QoS) peut s'assurer de la bonne livraison des messages si celle-ci est critique :

— **Niveau 0 - At most once :** message envoyé une seule fois, sans garantie de réception

— **Niveau 1 - At least once :** message envoyé au moins une fois, jusqu'à ce que le broker MQTT en accuse sa réception

— **Niveau 2 - Exactly once :** message envoyé jusqu'à ce que sa transmission soit confirmée mais de sorte à éviter les doublons

Deuxièmement, comme MQTT est largement répandu, son intégration à une application devient facile peu importe le langage de programmation employé.

2.3.2. Bluetooth direct

Avec MQTT, les messages BLE doivent être traduits en messages MQTT puis envoyés au broker et vice-versa. Il est également possible d'utiliser des librairies qui s'appuient directement sur la connectique Bluetooth d'un appareil pour établir une connection avec un objet connecté et écrire/lire la valeur des caractéristiques sans passer par un quelconque intermédiaire.

3
Expérimentations

3.1. Matériel utilisé 11
3.1.1. Objets connectés 11
3.1.2. Points d'accès 12
3.1.3. Langages utilisés 12
3.2. Premiers pas : noble-device 13
3.2.1. Architecture 13
3.2.2. Fonctionnalités 13
3.2.3. Limitations 13
3.3. Raspberry Pi : EspruinoHub, MQTT 14
3.3.1. Architecture 14
3.3.2. Fonctionnement 14
3.3.3. Limitations 14
3.4. Raspberry Pi : Node-RED, EspruinoHub, MQTT 15
3.4.1. Architecture 15
3.4.2. Fonctionnement 15
3.4.3. Limitations 16
3.5. ESP32 : esp32-ble2mqtt, MQTT 16
3.5.1. Architecture 16
3.5.2. Fonctionnement 17
3.5.3. Limitations 17
3.6. ESP32 : esp32-ble2mqtt, Mozilla Gateway 17
3.6.1. L'implication de Mozilla dans l'IoT 17
3.6.2. Intégration du Thingy :52 au Mozilla Gateway 19
3.6.3. Limitations 20
3.7. Web Bluetooth API 21
3.7.1. Architecture 21
3.7.2. Fonctionnement 21
3.7.3. Implémentation 21
3.7.4. Limitations 22
3.8. Conclusion 22

3.1. Matériel utilisé

3.1.1. Objets connectés

Nordic Thingy:52

Extrêmement polyvalent, le Thingy:52 de Nordic se révèle être l'appareil parfait pour le développement IoT. En effet, son plus grand avantage est d'être un tout-en-un au niveau de ses capteurs et actuateurs (température, pression, CO_2, humidité, lumière, accéléromètre, bouton, LED, microphone, haut-parleur, etc.), ce qui offre au développeur tout le loisir d'élaborer des scénarios relativement complexes à l'aide d'un unique objet.

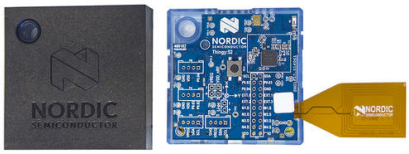


Figure 3.1. – Nordic Thingy:52

Magic Blue : ampoule LED

Acquise à moindre coût sur internet, cette ampoule possède toutes les fonctionnalités d'ampoules de grandes marques, tout en respectant le standard GATT. Il est bien évidemment possible de modifier sa couleur de façon constante, mais aussi de naviguer parmi les vingt réglages prédéfinis.

Figure 3.2. – Ampoule LED Magic Blue

iTAG

Pensé à la base pour retrouver ses clés puisqu'il sonne lorsqu'il perd la connexion Bluetooth, l'iTAG est un petit objet peu onéreux que l'on peut faire sonner comme une alarme.

Figure 3.3. – iTAG

3.1.2. Points d'accès

Dans un contexte réel où la connexion Bluetooth doit être constamment accessible, il est intéressant d'examiner des solutions peu énergivores et bon marché comme points d'accès. En effet, la faible consommation de ces appareils permet un fonctionnement constant, tandis que le prix rend possible l'installation de plusieurs d'entre-eux afin de garantir une connexion stable sur un large périmètre.

Raspberry Pi 3 Model B

Le Raspberry Pi 3 Model B est un mini-ordinateur embarquant notamment 1 Go de RAM et un processeur quad-core 1.2 GHz. Il supporte le Wi-Fi et le BLE. Coté logiciel, *EspruinoHub* [1] fera office de passerelle BLE vers MQTT et lancera un broker MQTT directement sur le Raspberry Pi.

Figure 3.4. – Raspberry Pi 3 Model B

ESP32

Moins puissant que le Raspberry Pi et basé sur un microcontrôleur, l'ESP32 présente une connectivité identique en matière de Wi-Fi et de BLE. Il sera flashé avec *esp32-ble2mqtt* [2] qui servira de pont BLE vers MQTT. Contrairement à *EspruinoHub*, *esp32-ble2mqtt* nécessite la programmation d'un broker MQTT annexe tournant sur une autre machine.

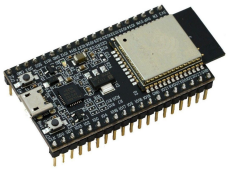


Figure 3.5. – ESP32

3.1.3. Langages utilisés

Référence en programmation web, les expérimentations s'effectueront en JavaScript (côté client, et également côté serveur à l'aide de Node.js). Ce choix s'appuie sur la grande variété de librairies et frameworks consacrés aux objets connectés et aux applications web en temps réel disponibles pour cet environnement.

[1] https ://github.com/espruino/EspruinoHub
[2] https ://github.com/shmuelzon/esp32-ble2mqtt

3.2. Premiers pas : noble-device

3.2.1. Architecture

Figure 3.6. – Architecture - Noble-device

3.2.2. Fonctionnalités

En guise d'introduction au monde des objets connectés et de JavaScript, un service web RESTful permettant d'obtenir les valeurs des capteurs d'un Thingy en JSON a été conçu. Ce service intègre une base de données clé-valeur (Redis) qui permet d'avoir un historique des données relevées. Grâce à cela, le service web a un potentiel de développement intéressant. L'objectif étant ici la découverte de ces technologies et de la programmation asynchrone, l'aspect Bluetooth n'a pas été approfondi et repose entièrement sur la librairie fournie par Nordic, basée elle-même sur *noble-device*[3].

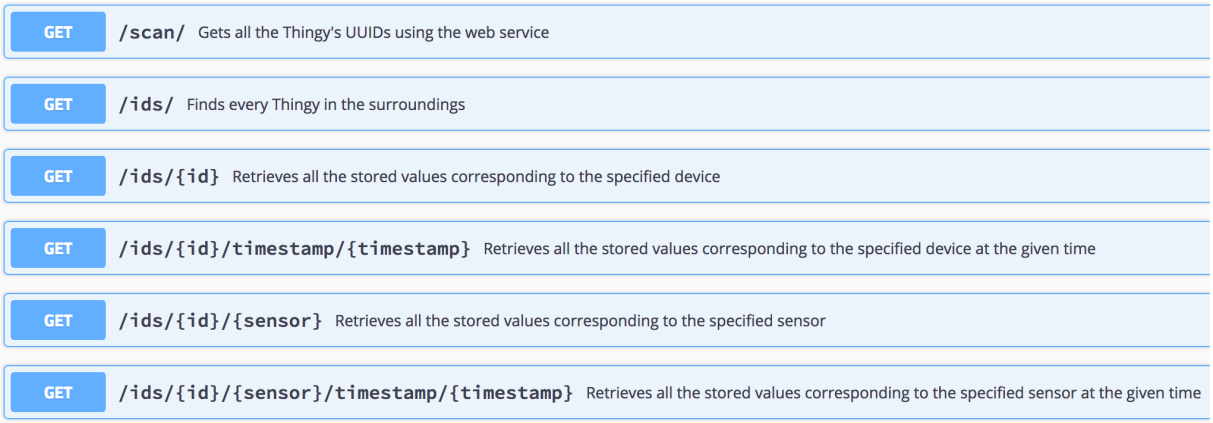


Figure 3.7. – Fonctionnalités - SwaggerUI


```
temperature:
    Fri Jul 20 2018 15:17:09 GMT+0200 (CEST):    "27.87"
    Fri Jul 20 2018 15:17:10 GMT+0200 (CEST):    "27.99"
    Fri Jul 20 2018 15:17:11 GMT+0200 (CEST):    "28.12"
```


Figure 3.8. – Valeurs du capteur de température d'un Thingy. JSON.

3.2.3. Limitations

Ce service reste très basique. Il est entièrement réservé au Thingy et nécessite quelques perfectionnements pour le rendre réellement utile. En revanche, sa conception a rempli l'objectif visé : se familiariser avec les technologies utilisées.

[3]https ://github.com/noble/noble-device

3.3. Raspberry Pi : EspruinoHub, MQTT

3.3.1. Architecture

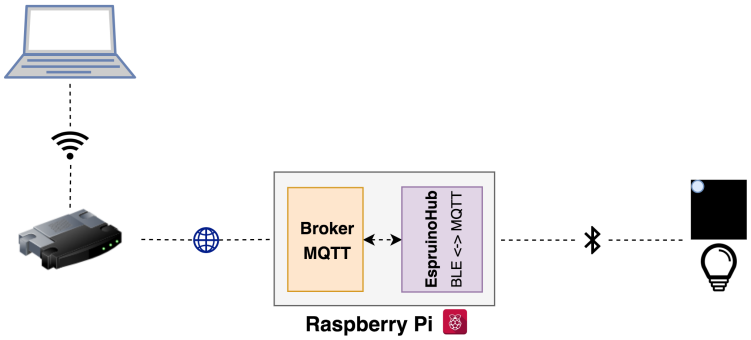


Figure 3.9. – Architecture - Raspberry Pi - EspruinoHub

3.3.2. Fonctionnement

Un Raspberry Pi joue le rôle de point d'accès central. Les objets connectés communiquent directement avec lui via BLE. *EspruinoHub*[4] sert de traducteur BLE vers messages MQTT et lance également un broker MQTT local (qui pourrait tout à fait être externalisé sur une autre machine). Ainsi, les clients se connectent au Raspberry Pi comme s'ils se connectaient à n'importe quel broker MQTT (à l'aide de Mosquitto par exemple).

EspruinoHub propose un format facile à appréhender comme par exemple `/ble/advertise/DEVICE` ou `/ble/read/DEVICE/SERVICE/CHARACTERISTIC` pour les messages "advertise" et "read" respectivement. Pour rappel, les messages "advertise" servent à la détection d'objets Bluetooth environnants et contiennent notamment le nom de l'appareil et l'intensité du signal.

3.3.3. Limitations

EspruinoHub a l'avantage d'être une solution tout-en-un. En effet, elle joue à la fois le rôle de gateway BLE vers MQTT et de broker MQTT (de base, un serveur tourne en arrière-plan). Les tests ont montré que la récupération de données se faisait sans difficulté. En revanche, l'écriture sur certaines caractéristiques (couleur de l'ampoule notamment) ne provoquait aucune réaction. De plus, la mise en place initiale requiert de nombreux réglages. Il est d'ailleurs nécessaire de reparamétrer certaines options liées la connexion Bluetooth après chaque redémarrage de l'appareil. Ces désagréments, causés par la partie logicielle et non par le Raspberry Pi qui reste attrayant pour des projets en IoT, limitent quelque peu les possibilités d'utilisation à plus grande échelle.

[4]https ://github.com/espruino/EspruinoHub

3.4. Raspberry Pi : Node-RED, EspruinoHub, MQTT

3.4.1. Architecture

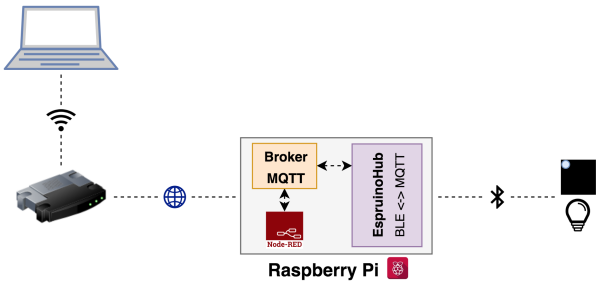


Figure 3.10. – Architecture - Raspberry Pi - Node-RED

3.4.2. Fonctionnement

Construit sur une base de Node.js, *Node-RED*[5] est une interface de programmation sous forme de flux. Des noeuds d'entrée et de sortie permettent de construire des scénarios en toute sorte de façon graphique, en offrant également un éditeur de fonctions JavaScript pour plus de personnalisation. Comme illustré sur la figure 3.10, *EspruinoHub* transforme les messages BLE en messages MQTT et joue également le rôle de broker (comme décrit dans la section 3.3). *Node-RED* se sert ensuite des messages MQTT pour ses scénarios. Ici, le scénario se concentre exclusivement sur les messages MQTT diffusés localement puisque *Node-RED* et *EspruinoHub* fonctionnent tous deux sur le Raspberry Pi.

Exemple de flux

La figure 3.11 présente un cas concret. Le flux débute dans la partie 1 où un noeud MQTT souscrit au topic guettant la présence d'un appareil particulier (dans cet exemple, il s'agit d'un Thingy) grâce aux messages "advertise" de celui-ci. Dès qu'il est détecté, un "1" est publié dans le topic correspondant, ce qui cause la souscription aux notifications de plusieurs de ses caractéristiques (température, pression, humidité, CO_2 et lumière). Après cela, les noeuds MQTT de la partie 2 prennent le relais en récupérant toutes les valeurs publiées par ces caractéristiques afin de les afficher dans la partie 3, la console de *Node-RED*. Cette base ouvre d'autres portes comme le transfert de ces données vers un service web via websocket ou HTTP, ou un traitement direct au sein même de *Node-RED*, en généralisant la détection d'objets connectés à d'autres Thingys et d'autres objets.

[5]https ://nodered.org/

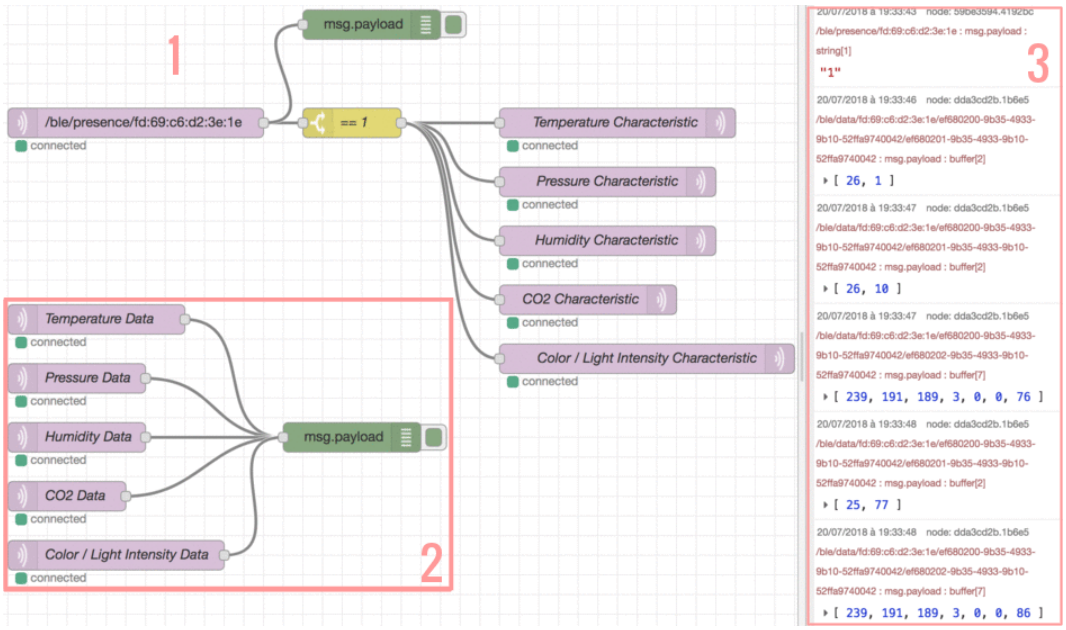


Figure 3.11. – Node-RED : exemple de flux

3.4.3. Limitations

Node-RED est un outil facile à prendre en main. L'aspect graphique et la grande quantité de noeuds prédéfinis ouvrent la création de scénarios à un public profane. Ainsi, il devient aisé de mélanger objets connectés, sites web et réseaux sociaux au sein d'un seul éditeur. Les développeurs plus aguerris ne sont pas en reste puisque le service permet la conception de noeuds personnalisables grâce à l'intégration de code JavaScript notamment. Afin de prévenir toute confusion causée par la démultiplication de noeuds dans le cas de projets plus complexes, *Node-RED* propose également de grouper des noeuds en sous-flux, représentés ensuite par un simple noeud. Il s'agit donc un outil puissant. En revanche, les limitations liées au Raspberry Pi et à *EspruinoHub* (voir 3.3.3) s'appliquent toujours.

3.5. ESP32 : esp32-ble2mqtt, MQTT

3.5.1. Architecture

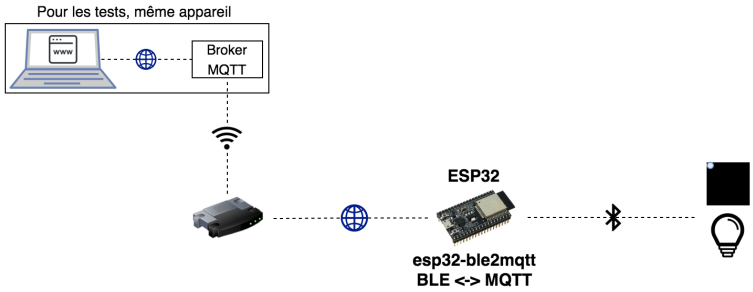


Figure 3.12. – Architecture - ESP32 : esp32-ble2mqtt

3.5.2. Fonctionnement

esp32-ble2mqtt[6] s'occupe de transformer les informations obtenues via BLE en messages MQTT qui sont ensuite redirigés vers un broker externe. Ici, le broker MQTT sera une application Node.js utilisant le package MOSCA, qui fonctionnera sur l'ordinateur principal (voir figure 3.12). Par défaut, *esp32-ble2mqtt* essaie de se connecter à tous les appareils BLE à proximité, en souscrivant à toutes les caractéristiques de tous leurs services. Or, cela peut générer de l'instabilité. De ce fait, il convient de les trier à l'aide d'une liste blanche ou noire. Enfin, *esp32-ble2mqtt* permet de remplacer les UUIDs à 16 ou 128 bits dans les topics MQTT par des chaînes de caractères (plus lisibles) qui doivent bien évidemment être définies en amont (figure 3.13).

```
I (13888) BLE2MQTT: Publishing: fd:69:c6:d2:3e:1e/Thingy Environment Service/Thingy Temperature Characteristic = 24,0
I (13918) BLE2MQTT: Publishing: fd:69:c6:d2:3e:1e/Thingy Environment Service/Thingy Pressure Characteristic = 930,22
I (13948) BLE2MQTT: Publishing: fd:69:c6:d2:3e:1e/Thingy Environment Service/Thingy Humidity Characteristic = 53
I (14188) BLE2MQTT: Publishing: fd:69:c6:d2:3e:1e/Thingy Environment Service/Thingy Light Intensity Characteristic = 2877,3152,3288,493
I (14878) BLE2MQTT: Publishing: fd:69:c6:d2:3e:1e/Thingy Environment Service/Thingy Temperature Characteristic = 23,61
I (14908) BLE2MQTT: Publishing: fd:69:c6:d2:3e:1e/Thingy Environment Service/Thingy Pressure Characteristic = 930,27
I (14938) BLE2MQTT: Publishing: fd:69:c6:d2:3e:1e/Thingy Environment Service/Thingy Humidity Characteristic = 48
I (14978) BLE2MQTT: Publishing: fd:69:c6:d2:3e:1e/Thingy Environment Service/Thingy Air Quality Characteristic = 0,0
I (15178) BLE2MQTT: Publishing: fd:69:c6:d2:3e:1e/Thingy Environment Service/Thingy Light Intensity Characteristic = 2875,3325,3573,510
```

Figure 3.13. – ESP32 : esp32-ble2mqtt - Topics MQTT et valeurs

3.5.3. Limitations

esp32-ble2mqtt est satisfaisant pour la souscription à plusieurs caractéristiques et la réception des valeurs lorsque le nombre d'appareils est faible. Lorsqu'il devient plus important (3 et plus), les crashes intempestifs deviennent monnaie courante. L'écriture sur une caractéristique fonctionne quant à elle de façon aléatoire. La couleur de l'ampoule Magic Blue a pu être réglée avec succès mais une seule fois, après quoi un reset de l'ESP32 était requis pour que la commande fasse à nouveau effet. Grâce à ce projet open source prometteur, il est possible d'envisager l'utilisation d'un ESP32 comme point d'accès BLE. Malheureusement, il manque encore de maturité pour être utilisé dans un scénario plus conséquent.

3.6. ESP32 : esp32-ble2mqtt, Mozilla Gateway

3.6.1. L'implication de Mozilla dans l'IoT

W3C WoT Thing Description et Mozilla Web Thing API

A l'heure actuelle, deux standards majeurs existent pour la définition et la description d'objets physiques. Le premier, géré par le World Wide Web Consortium (W3C), se dénomme WoT Thing Description. Le deuxième est la Web Thing API de Mozilla. Logiquement, certaines différences existent entre les deux spécifications. Les listings 3.1 et 3.2 démontrent les différences dans la description d'un objet physique. Par exemple, alors que la solution du W3C place les propriétés, actions et événements comme sous-classes au sein de la ressource généraliste "interaction", celle de Mozilla structure autrement les informations en séparant les propriétés, actions et événements, ce qui la rend plus lisible

[6]https ://github.com/shmuelzon/esp32-ble2mqtt

au premier abord. Naturellement, la Web Thing API est au centre du projet de Mozilla décrit dans cette section.

```
{
  "@context": ["https://w3c.github.io/wot/w3c-wot-td-context.jsonld",
               "https://w3c.github.io/wot/w3c-wot-common-context.jsonld"
               ],
  "@type": ["Sensor"],
  "name": "myTempSensor",
  "base" : "coap:///www.example.com:5683/temp/",
  "interaction": [
    {
      "@type": ["Property","Temperature"],
      "reference": "threshold",
      "name": "myTemp",
      "schema": {
        "type": "number"
      },
      "writable": false,
      "observable": true,
      "form": [{
          "href" : "val",
          "mediaType": "application/json"
      }]
    },
    {
      "@type": ["Property","Temperature"],
      "name": "myThreshold",
      "schema": {
        "type": "number"
      },
      "writable": true,
         "form": [{
          "href" : "thr",
          "mediaType": "application/json"
      }]
    },
    {
      "@type": ["Event"],
      "schema": { "type": "number" },
      "name": "myChange",
      "property": "temp",
      "form": [{
          "href" : "val/changed",
          "mediaType": "application/json"
      }]
    },
    {
      "@type": ["Event"],
      "schema": { "type": "number" },
      "name": "myWarning",
      "form": [{
          "href" : "val/high",
          "mediaType": "application/json"
      }]
    }
  ]
}
```

Listing 3.1 – WoT Thing Description. JSON. [15]

```
{
  "name":"WoT Pi",
  "description": "A WoT-connected Raspberry Pi",
  "properties": {
    "temperature": {
      "label": "Temperature",
      "type": "number",
      "unit": "celsius",
      "description": "An ambient temperature sensor",
      "href": "/things/pi/properties/temperature"
    },
    "humidity": {
      "label": "Humidity",
      "type": "number",
      "unit": "percent",
      "href": "/things/pi/properties/humidity"
    },
    "led": {
      "label": "LED",
      "type": "boolean",
      "description": "A red LED",
      "href": "/things/pi/properties/led"
    }
  },
  "actions": {
    "reboot": {
      "label": "Reboot",
      "description": "Reboot the device"
    }
  },
  "events": {
    "reboot": {
      "description": "Going down for reboot"
    }
  },
  "links": [
    {
      "rel": "properties",
      "href": "/things/pi/properties"
    },
    {
      "rel": "actions",
      "href": "/things/pi/actions"
    },
    {
      "rel": "events",
      "href": "/things/pi/events"
    },
    {
      "rel": "alternate",
      "href": "wss://mywebthingserver.com/things/pi"
    },
    {
      "rel": "alternate",
      "mediaType": "text/html",
      "href": "/things/pi"
    }
  ]
}
```

Listing 3.2 – Web Thing API. JSON. [18]

"Project Things" : Things Gateway

Mozilla a l'intention d'uniformiser l'interaction entre objets connectés grâce à son projet orienté IoT. La figure 3.14 décrit son architecture. L'idée de ce gateway est de centraliser la gestion d'objets connectés divers dans une unique interface qui propose d'ailleurs un éditeur de règles conditionnelles if-then et un outil de gestion des appareils selon un plan d'étage, tous deux encore peu développés.

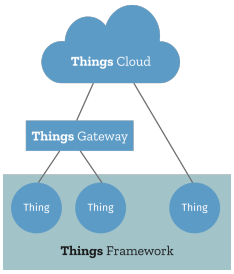


Figure 3.14. – Mozilla IoT

3.6.2. Intégration du Thingy:52 au Mozilla Gateway

La figure 3.15 expose l'architecture employée. Pour des raisons pratiques, les éléments logiciels fonctionnent tous sur le même ordinateur. Bien entendu, ce n'est pas obligatoire. Ces éléments peuvent être répartis sur des appareils différents (à l'exception du gateway et de l'adaptateur qui forment une paire).

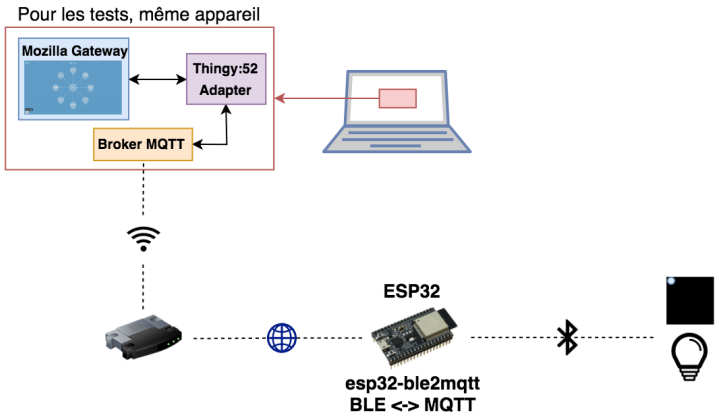


Figure 3.15. – Architecture - Mozilla Gateway

Pour qu'un objet puisse se connecter au Things Gateway, un adaptateur, portion de code redéfinissant des fonctions de la Web Thing API, doit être programmé. Mozilla compte dès lors sur la communauté de développeurs pour étendre la palette d'appareils supportés. En partant d'un exemple fictif fourni par Mozilla, les classes suivantes et certaines de leurs fonctions doivent subir des modifications :

— **Device :** L'objet Device représente l'objet connecté ainsi que les propriétés et actions associées. Une bonne pratique est de le représenter sous forme d'une constante JSON qui respecte le standard de Mozilla et qui contient notamment le nom, le type, une description et les propriétés de l'appareil.

— **Property :** Une propriété est une caractéristique de l'objet connecté (température, etc.).

— **Adapter :** L'adaptateur fait le lien entre le gateway et l'objet connecté. Il gère les processus d'appairage, d'ajout et de suppression d'objets.

Résultat

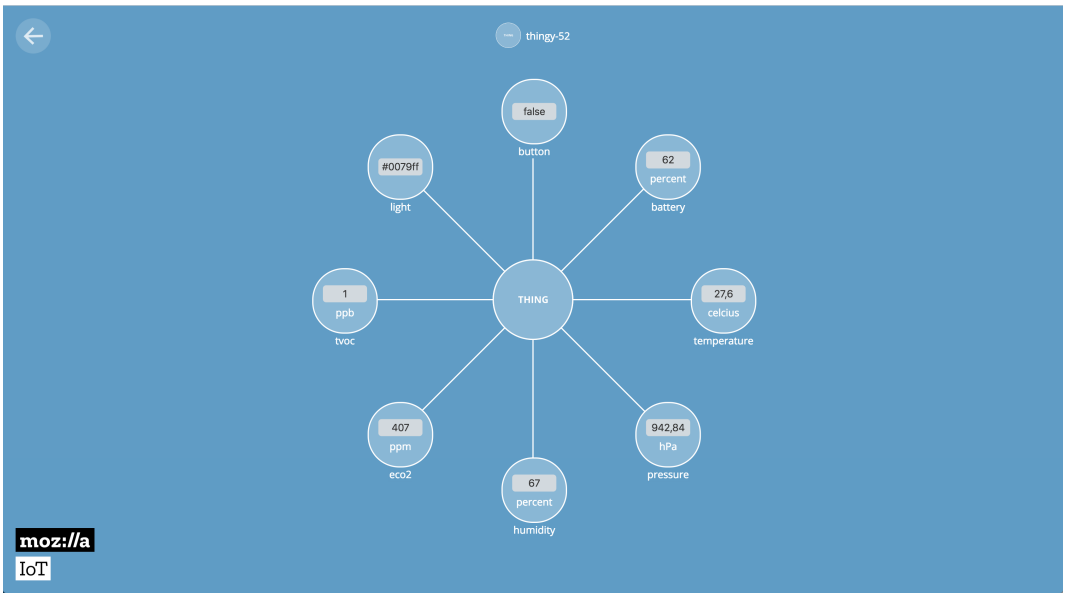


Figure 3.16. – Mozilla Gateway - Thingy:52

Dans cet exemple, l'ajout d'un Thingy:52 engendre la souscription à certaines caractéristiques dont les valeurs sont récupérées via MQTT et actualisées en temps réel grâce à l'adaptateur conçu dans le cadre de ce travail (voir figure 3.15). La figure 3.16 expose l'apparence d'un Thingy dans l'interface.

3.6.3. Limitations

L'idée globale de Mozilla de réunir les objets connectés au sein d'une interface web est ingénieuse. A l'heure actuelle, le projet est à ses débuts. De ce fait, les fonctionnalités comme la création de scénarios if-then sont encore limitées. Grâce à la communauté, de plus en plus d'appareils sont supportés même s'ils restent pour l'instant peu nombreux. Une idée intéressante serait de définir un adaptateur générique pour certains types d'appareils afin que le développeur puisse emprunter une approche déclarative pour représenter un nouvel appareil (sous forme d'un fichier JSON par exemple), mais l'implémentation actuelle ne le permet pas, en sachant qu'il faudrait également proposer une façon unique de communiquer avec les objets BLE. L'exemple de ce travail a utilisé la combinaison ESP32 et MQTT, mais d'autres moyens plus stables et évolutifs existent.

3.7. Web Bluetooth API

3.7.1. Architecture

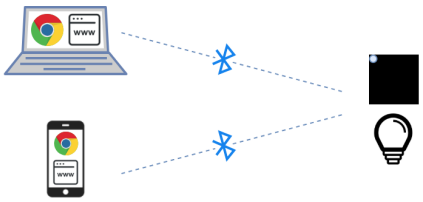


Figure 3.17. – Architecture - Web Bluetooth API

3.7.2. Fonctionnement

Cette API JavaScript pour navigateurs, actuellement compatible avec certains d'entre-eux uniquement (Google Chrome notamment), utilise le hardware Bluetooth intégré à l'appareil pour établir une connexion avec des objets BLE via le protocole GATT. Par conséquent, la connexion est indépendante d'intermédiaires comme dans les exemples explicités précédemment, ce qui élargit le spectre d'utilisateurs potentiels.

3.7.3. Implémentation

Notifications

L'API se base sur les "promises" afin de garantir l'ordre d'exécution des instructions. Voici un exemple généraliste de l'utilisation de l'API pour souscrire aux notifications d'une caractéristique d'un Thingy :

```
1  // détection des Thingys à proximité et association
2  navigator.bluetooth.requestDevice({filters: [{namePrefix:"Thingy"}],
3                                     optionalServices: [
4                                       "ef680200-9b35-4933-9b10-52ffa9740042",
5                                       "ef680300-9b35-4933-9b10-52ffa9740042",
6                                       "ef680500-9b35-4933-9b10-52ffa9740042"
7                                     ]
8  })
9  .then(device => {
10   // établissement de la connexion
11   return device.gatt.connect();
12 })
13 .then(server => {
14   // récupération du service à l'aide de son UUID
15   return server.getPrimaryService(serviceUUID);
16 })
17 .then(service => {
18   // récupération de la caractéristique à l'aide de son UUID
19   return service.getCharacteristic(characteristicUUID);
20 })
21 .then(characteristic => {
22   // ici, caractéristique de type "Notify"
```

```
23    return myCharacteristic.startNotifications().then(_ => {
24              myCharacteristic.addEventListener('characteristicvaluechanged',
                    handleEvent(event));
25    });
26  })
27  .catch(error => {
28    handleError(error);
29  });
```

Listing 3.3 – Connexion et souscription aux notifications d'une caractéristique d'un Thingy. JavaScript.

Dans un cas comme le Thingy où de nombreuses caractéristiques entrent en jeu, il est judicieux de spécifier à la fonction de gestion d'événements de la ligne 24 à quelle caractéristique l'on souscrit, ou d'utiliser une fonction spécifique par caractéristique.

Ecriture

De façon analogue, il est possible de modifier la valeur d'une caractéristique :

```
1  [...]
2  .then(characteristic => {
3    return characteristic.writeValue(Uint8Array.of(86,255,0,0,0,240,170));
4  })
5  [...]
```

Listing 3.4 – Changement de la couleur d'une ampoule Magic Blue en rouge. JavaScript.

Tableau de bord

Cette solution a été employée dans l'élaboration d'un tableau de bord supportant tous les objets connectés cités dans ce travail afin d'écrire ou de souscrire à toutes les caractéristiques de ces derniers. Il intégre également un service de météo externe se basant sur la localisation GPS (afin de comparer les valeurs réelles avec les valeurs relevées par les Thingys) et une interface de gestion de scénarios basiques (voir figure 3.18).

3.7.4. Limitations

Cette API a l'avantage d'être stable et facile à manier puisqu'elle ne demande qu'un navigateur compatible pour fonctionner. De ce fait, une solution combinant web et objets connectés construite sur l'API Web Bluetooth a le potentiel de toucher un grand nombre d'utilisateurs, bien que la compatibilité encore limitée à certains navigateurs le réduise.

3.8. Conclusion

Chaque architecture présente des points positifs et négatifs. Le choix de la meilleure solution dépendra donc de l'importance accordée à chaque aspect (stabilité, accessibilité, matériel, etc.). Parmi toutes les solutions étudiées, l'API Web Bluetooth se démarque par sa stabilité et sa facilité d'utilisation (voir section 3.7 pour de plus amples détails). Cette

solution sera donc retenue dans l'implémentation d'un scénario plus global présenté dans le chapitre 4.

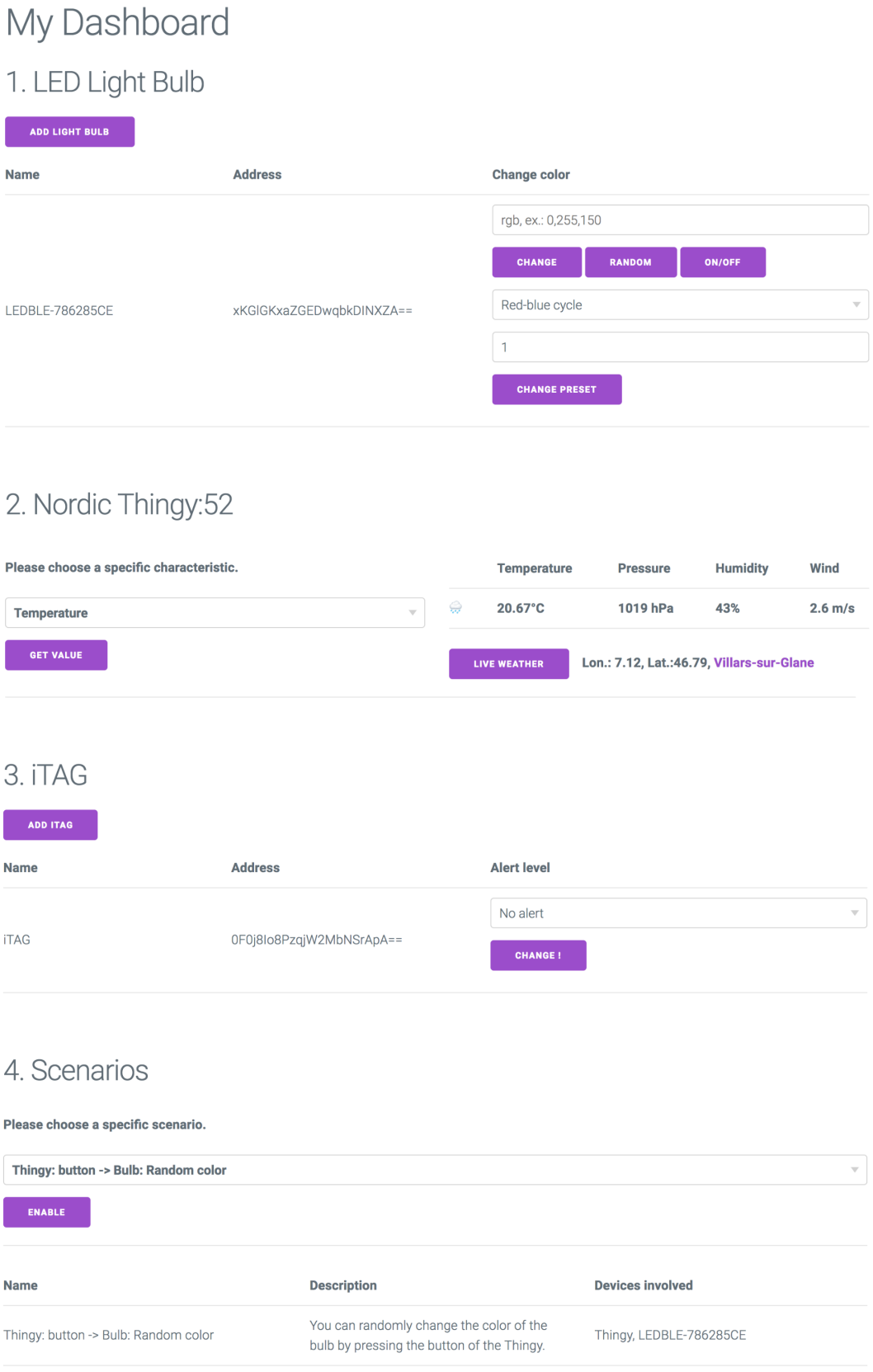


Figure 3.18. – My Dashboard - Tableau de bord basé sur l'API Web Bluetooth

4

Application des expérimentations : Objets BLE et Spotify

4.1. Motivation 24
4.2. Concept 24
4.2.1. Connexion via Spotify : Fonctionnalités 25
4.2.2. Connexion à une session en cours : Fonctionnalités 28
4.3. Spotify Web API 28
4.3.1. Connexion et vie privée 28
4.3.2. spotify-web-api-js 29
4.4. Implémentation 29
4.4.1. Code source 29
4.4.2. Lecteur musical 29
4.4.3. Mood 30
4.4.4. Party 30
4.5. Extensions et améliorations 31
4.6. Démonstration 31

4.1. Motivation

Le chapitre précédent a permis de parcourir différentes manières d'interagir avec des objets intelligents. Or, cette interaction ne prend un sens que si elle s'insère dans un scénario plus global où elle apporte un plus à l'expérience utilisateur. L'objectif de ce dernier chapitre est de mettre en exergue l'intérêt des objets connectés dans une application web concrète qui les voit comme de simples outils et non comme sujets principaux.

4.2. Concept

L'utilisateur a le choix de se connecter à l'aide de son compte Spotify ou de rejoindre une session déjà en cours à l'aide d'un code à 6 chiffres propre à la session (voir figure 4.1).

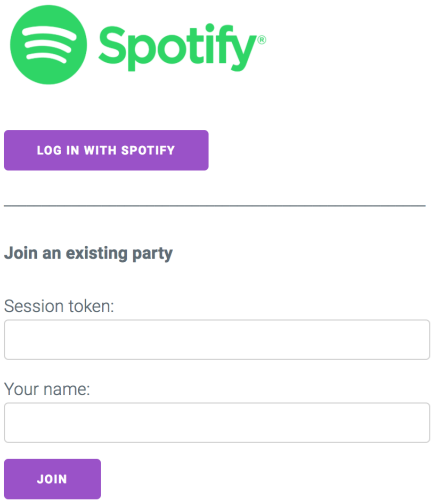


Figure 4.1. – Interface - Connexion

4.2.1. Connexion via Spotify : Fonctionnalités

Eléments visuels

L'interface est divisée en deux parties. La première est consacrée au lecteur musical où l'utilisateur verra en temps réel la chanson en cours de lecture, l'état du mode Mood et du mode Party (figure 4.2).

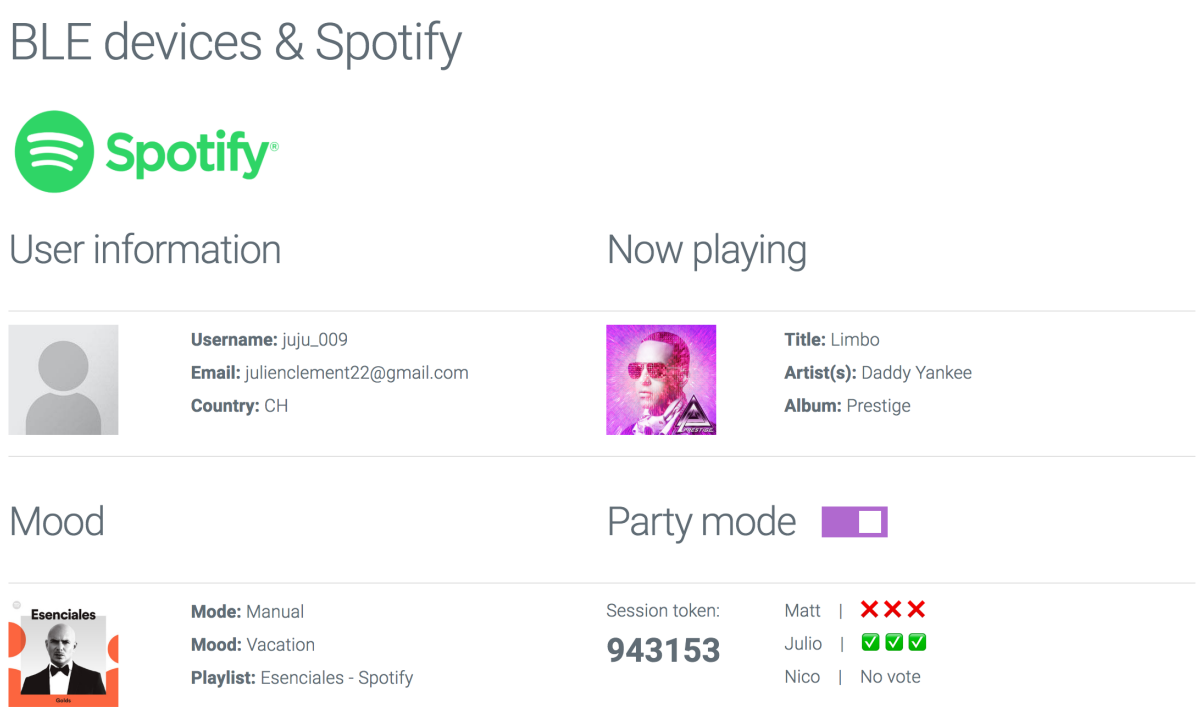


Figure 4.2. – Interface - Partie Spotify

La deuxième partie se concentre quant à elle sur les objets connectés et permet d'associer jusqu'à 2 Thingys et des ampoules Magic Blue (figure 4.3).

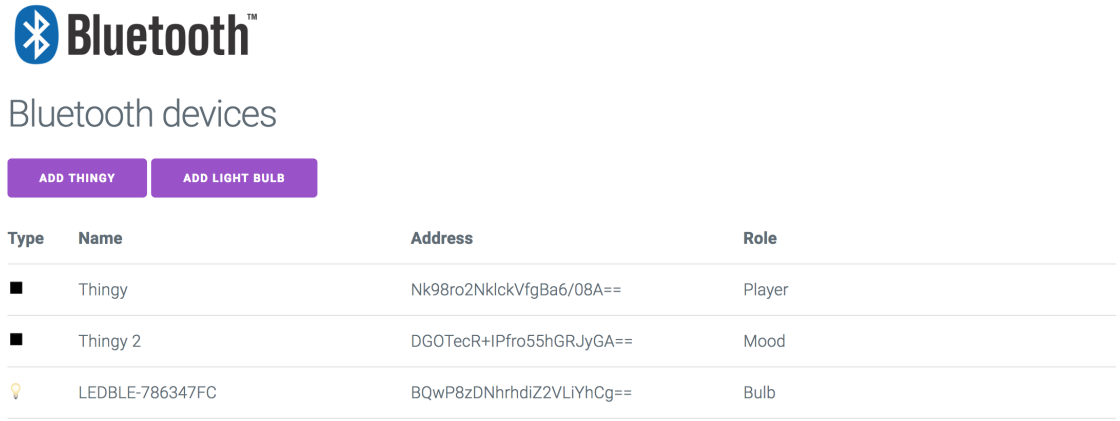


Figure 4.3. – Interface - Partie Bluetooth

Premier Thingy : Player

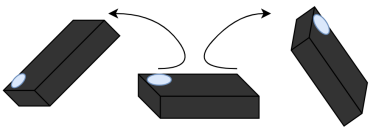
Le premier Thingy offre à l'utilisateur la possibilité d'influer sur la lecture musicale en cours, qu'elle se passe sur le même appareil ou sur un autre (pour autant qu'il s'agisse du même compte Spotify). Afin de profiter de toutes les fonctionnalités, il est nécessaire de posséder un compte Spotify Premium.

BOUTON

- **Appui simple :** Play/Pause
- **Appui long :** Ajout de la chanson en cours de lecture aux chansons sauvegardées

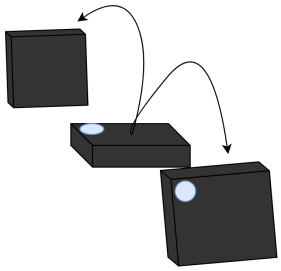
LED

- **Rouge :** Lecture en pause
- **Bleue :** Lecture en cours



ACCELEROMETRE : mouvement latéral

- **Vers la gauche :** Chanson précédente
- **Vers la droite :** Chanson suivante



ACCELEROMETRE : mouvement vertical

- **Vers l'avant :** Avance rapide
- **Vers l'arrière :** Retour rapide

Deuxième Thingy : Mode Mood

Le mode Mood est un système automatisant les choix musicaux et les choix d'ambiance (ampoules) en fonction de différents facteurs. Comme illustré sur la figure 4.4, ce mode est divisé en 3 états différents :

- **Manuel :** Une playlist et un preset d'ampoule sont définis pour chaque humeur (Sunny, Dream, Sad, Sport motivation, Angry, Focus, Nostalgia, Vacation, Party, Classical). Un appui simple sur le bouton du Thingy permet de passer à l'humeur suivante.
- **Automatique :** La meilleure playlist est choisie en fonction de l'heure de la journée et du temps actuel.
- **Désactivé :** Etat par défaut. Si l'utilisateur revient dans cet état après être passé par les deux modes précédents, la lecture de la playlist qu'il écoutait avant d'entrer dans le mode Mood sera relancée. Un appui simple sur le bouton du Thingy passe au preset d'ampoule suivant.

Le passage d'un état au suivant s'effectue à l'aide d'un appui long sur le bouton du Thingy. Evidemment, les actions du premier Thingy fonctionnent toujours.

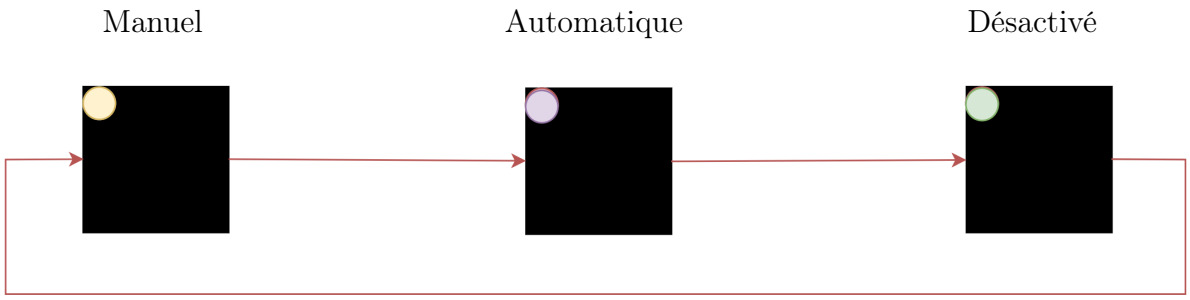


Figure 4.4. – Mode Mood - Transitions

Ampoules LED

Comme décrit précédemment, une ampoule ajoutée aux objets BLE connectés changera de couleur selon le mode Mood et les actions de l'utilisateur (appui simple sur le bouton du deuxième Thingy lorsque le mode Mood est désactivé par exemple, qui aura pour effet de naviguer parmi la vingtaine de réglages supportés).

Mode Party

Le mode Party intègre une dimension sociale au projet. Chaque session utilisant un compte Spotify comme moyen de connexion possède un token (code à 6 caractères) propre à elle.

Si le mode Party est activé pour une session, toute personne possédant le token peut rejoindre la fête correspondante et connecter son Thingy afin de donner son appréciation quant à la chanson en cours de lecture (voir sous-section 4.2.2). Si l'hôte de la fête ne désire pas que ses invités lui communiquent leur avis, il peut à tout moment désactiver le mode Party. Dans ce cas, plus personne ne pourra rejoindre la fête et aucun vote ne sera comptabilisé.

4.2.2. Connexion à une session en cours : Fonctionnalités

Eléments visuels

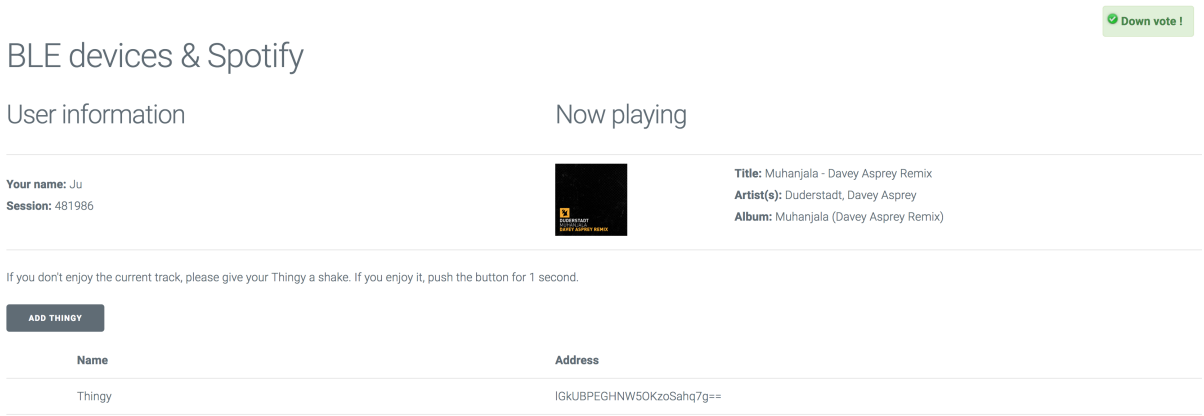


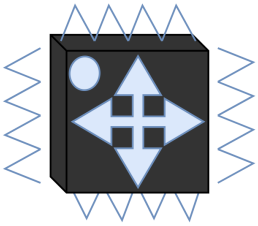
Figure 4.5. – Interface - Mode Party

Thingy

Le Thingy devient un outil de vote pour les invités, ce qui permet à l'hôte d'obtenir un feedback instantané et de satisfaire au mieux ces derniers. L'hôte de la fête garde toutefois un contrôle absolu sur la musique jouée.

BOUTON

— **Appui long :** Vote positif pour la chanson en cours de lecture



ACCELEROMETRE

— **Secousse :** Vote négatif pour la chanson en cours de lecture

4.3. Spotify Web API

4.3.1. Connexion et vie privée

La connexion à Spotify depuis une application externe s'effectue grâce au protocole OAuth 2.0. Les détails du processus de connexion n'étant pas pertinents dans le cadre de ce travail, ils ne seront pas détaillés ici, mais des explications sont disponibles sur le site web de Spotify [1]. En revanche, il convient de préciser que chacune des fonctionnalités de l'API

[1] https ://developer.spotify.com/documentation/general/guides/authorization-guide/

requiert des autorisations spécifiques fournies par l'utilisateur. Par exemple, l'utilisateur doit accorder la permission "user-modify-playback-state" afin que l'application puisse passer à la chanson suivante. Les applications externes se servant de cette API ne sont donc intrusives que si le développeur le souhaite et si l'utilisateur l'accepte.

4.3.2. spotify-web-api-js

Il existe deux façons d'accéder aux fonctionnalités de cette API RESTful : en effectuant les requêtes à la main ou en se servant d'un wrapper. Pour des raisons de praticité, c'est la deuxième solution qui a été retenue. Les appels se feront uniquement côté client, situation parfaitement adaptée au wrapper JavaScript *spotify-web-api-js*[2].

4.4. Implémentation

4.4.1. Code source

Le code source complet du projet est disponible sur GitHub à l'adresse suivante : `https://github.com/ClementJu/BLE-devices-Spotify`

4.4.2. Lecteur musical

Certaines fonctionnalités sont des simples appels à l'API Spotify (voir listing 4.1).

```
1  [...]
2    spotifyApi.getMyCurrentPlaybackState()
3    .then(
4      function(data){
5        spotifyApi.addToMySavedTracks([data.item.id], function(error, data){console.log
             ("Track added to saved tracks.")});
6      }, function(err){console.log(err);}
7    );
8  [...]
```

Listing 4.1 – Ajout d'une chanson aux chansons sauvegardées. JavaScript.

Dans de tels cas, la difficulté réside entièrement dans la gestion des objets connectés puisque deux actions sont notamment liées au bouton du Thingy grâce à un appui long et un appui court. Il a donc fallu trouver un moyen de différenciation en sachant que la caractéristique correspondant au bouton du Thingy envoie les notifications suivantes (à chaque changement d'état uniquement) :

- **0x00 :** bouton relâché
- **0x01 :** bouton pressé

Un appui sur le bouton est donc vu comme l'enchaînement d'un appui et d'un relâchement. Dès lors, c'est la différence de temps entre ces deux événements qui définit la longueur de l'appui. L'événement lié à une notification contient un timestamp qui servira à cette fin (voir listing 4.2).

[2]https ://github.com/jmperez/spotify-web-api-js

```
[...]
          // appui sur le bouton
          if(event.target.value.getUint8(0) == 1){
            lastPressedFirstThingy = event.timeStamp;
          }
          // relachement du bouton
          else if(event.target.value.getUint8(0) == 0){
            // voir la différence entre le dernier appui et ce relâchement

            // appui long
            if(event.timeStamp - lastPressedFirstThingy > 300 &&
                lastPressedFirstThingy != 0){
              // --> ajouter la chanson aux chansons sauvegardées
            }
            // appui normal
            else{
              if(isPlaying){
                // --> mettre en pause
              } else {
                // --> lancer la lecture
              }
            }
            lastReleasedFirstThingy = event.timeStamp;
          }
[...]
```

Listing 4.2 – Gestion du bouton du Thingy. JavaScript.

Il arrive également que l'API ne fournisse pas de fonctionnalité prédéfinie qui réaliserait une action souhaitée, comme c'est par exemple le cas pour l'avance/retour rapide, où le bon fonctionnement repose sur la combinaison entre une bonne détection du mouvement de l'accéléromètre et une utilisation de plusieurs fonctions de l'API Spotify (récupération de l'état d'avancement de la chanson, incrémentation de la valeur, modification de l'état d'avancement avec la nouvelle valeur, gestion des cas limites).

4.4.3. Mood

Une playlist et un preset ont été choisis pour chacune des humeurs. Il suffit ensuite de jouer la bonne playlist suivant le mode et d'implémenter les différentes actions du Thingy selon le mode.

4.4.4. Party

Websockets

Grâce aux websockets, un client et un serveur peuvent communiquer de façon événementielle sans avoir à effectuer de requêtes HTTP. Puisque la communication est très fréquente dans ce cas-ci et que le nombre de clients peut rapidement croître, elles permettent une transmission d'informations rapide sans surcharger le serveur.

En pratique

Le serveur Node.js écoute activement sur un port prédéfini. Une fois connectés via Spotify ou un token, les clients se connectent via websocket au serveur Node.js et se voient automatiquement attribuer une socket spécifique, comparable à une porte d'entrée individuelle. Chaque socket possède un ID, ce qui permettra de différencier les clients.

Dans cette implémentation, le but était de décentraliser au maximum les calculs auprès des clients. Cela signifie qu'un client connecté grâce à un token connaît l'ID de la socket de l'hôte de sa fête, et qu'un hôte possède la liste des IDs des sockets des clients connectés à sa fête. Ainsi, le serveur joue simplement un rôle d'intermédiaire en redistribuant les messages à la bonne socket suivant l'ID qu'il reçoit et l'événement survenu.

```
1  [...]
2  // événement "vote_up"
3  socket.on("vote_up", function(data){
4  // si l'hôte auquel le client est lié a bien activé le mode Party
5    if(partyModeState.get(data.socket)){
6      // transmission du vote au bon hôte, infos: nom du client et ID de sa socket
7      // sockets: Map(SocketID, Socket)
8      sockets.get(data.socket).emit("vote_up", {name: data.name, socket: socket.id});
9    }
10   else{
11     // refus du vote si le mode Party est désactivé
12     socket.emit("party_mode_disabled");
13   }
14 });
15 [...]
```

Listing 4.3 – Gestion d'un vote positif côté serveur. Node.js.

Le serveur se charge également d'informer les parties prenantes lors de la déconnexion d'un hôte ou d'un client afin de fournir un service stable et fonctionnel en toute situation.

4.5. Extensions et améliorations

L'API Spotify offrant des options de recherche, il serait d'intéressant d'impliquer l'utilisateur dans la personnalisation du service : choix des playlists et des ambiances lumineuses pour les humeurs, utilisation de playlists créées par l'utilisateur, etc. De même, l'automatisation des choix musicaux et la sélection de la meilleure playlist selon des facteurs externes pourraient inclure les habitudes de l'utilisateur (emploi du temps, activité pratiquée, etc.).

Toujours en exploitant le côté recherche de l'API, l'interface pourrait proposer de jouer n'importe quelle chanson du catalogue Spotify. Grâce à cela, le mode Party se verrait rajouter un mode où l'hôte pourrait soumettre aux votes de ses invités plusieurs choix pour la prochaine chanson ou playlist.

4.6. Démonstration

Les fonctionnalités principales sont mises en pratique dans une vidéo de démonstration disponible à l'adresse suivante : `https://youtu.be/gWMGsb5V7BQ`

5 Conclusion

5.1. Synthèse

Les produits supportant le BLE ne manquent pas et les technologies pour les utiliser non plus. Cependant, il est évident qu'aucune d'entre-elles n'est actuellement parfaite et que tout choix imposera forcément des compromis. Parmi les solutions étudiées, la plus convaincante fut la Web Bluetooth API qui se démarque par sa stabilité et sa facilité d'utilisation une fois maîtrisée. Le point noir majeur réside actuellement en sa compatibilité limitée à certains navigateurs.

La facilité de création d'une interaction entre divers objets connectés varie au cas par cas. La condition sine qua non est qu'ils utilisent tous le protocole GATT. Ensuite, tout dépend de l'architecture choisie par le fabricant. Il existe ainsi des questions récurrentes pour chaque ajout d'appareil : quel est le format (Int, Uint, Float, 8 bits, 16 bits, 32 bits, etc.) choisi pour la lecture et l'écriture des données ? A quelle action correspondent les valeurs écrites ou lues pour des caractéristiques particulières comme une LED ? Si la documentation fournie est complète et ouverte aux développeurs, ces questions ne seront pas un obstacle. Dans le cas contraire, seules l'expérimentation et l'aide fournie par la communauté peuvent apporter ces réponses essentielles, sans quoi les recherches se révéleront rapidement chronophages.

Une fois ces détails techniques maîtrisés pour chaque objet, le développeur peut enfin laisser parler sa créativité. Tout ne devient qu'une question d'idées et de temps.

5.2. Evolution future

L'interaction homme-machine se développe et ne se limite plus à l'utilisateur devant son ordinateur. Les objets connectés ont la capacité d'ajouter une dimension supplémentaire à tout projet IT, à condition de choisir les bons. Ils sont, pour la plupart, composés de capteurs et d'actuateurs similaires. La différence doit ainsi se faire dans la partie logicielle et dans l'utilité des actions associées. Pour être intéressant à large échelle et pour le plus grand nombre, un objet doit impérativement faciliter la vie de l'utilisateur en réduisant le nombre d'actions nécessaires ou en offrant des fonctionnalités supplémentaires car les données seules ne suffisent pas. Elles doivent être traitées de façon convenable et compréhensive pour l'utilisateur final, sans quoi leur utilité sera remise en question. Peu de gens ont simplement envie de savoir que leur dernière séance de course à pied a duré

57 minutes et leur a permis de brûler 428 kcal. En revanche, la majorité s'intéressera à une description chiffrée de l'évolution de ses performances sur le long terme, en tenant compte de facteurs extérieurs comme la chaleur ou la pluie. Ainsi, deviendra leader du secteur celui qui combinera un écosystème complet et facile d'utilisation à des fonctionnalités innovantes et pertinentes pour l'utilisateur. L'adoption de cette technologie par le grand public passera inévitablement par soit une intercompatibilité entre les appareils et logiciels du marché, soit par la complétude de l'offre d'un fabricant qui transformera le milieu en quasi-monopole. Un point demeure certain, les objets connectés ont un potentiel incroyable qui ne demande qu'à être exploité.

A

Acronymes communs

API	Application Programming Interface
BLE	Bluetooth Low Energy
GATT	Generic Attribute Profile
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IoT	Internet of Things
JSON	JavaScript Object Notation
MQTT	Message Queuing Telemetry Transport
QoS	Quality of Service
REST	Representational State Transfer
TCP	Transmission Control Protocol
URI	Unified Resource Identifier
W3C	World Wide Web Consortium
WoT	Web of Things

Références

[1] Node.js Express Framework. https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm (dernière consultation le 21 fevrier 2018).

[2] Dominique D. Guinard and Vlad M. Trifa. *Building the Web of Things*. Manning, June 2016. 4, 5

[3] Reverse Engineering a Bluetooth Lightbulb | Uri Shaked | Medium. https://medium.com/@urish/reverse-engineering-a-bluetooth-lightbulb-56580fcb7546 (dernière consultation le 30 mai 2018).

[4] Getting started with MQTT. http://thejackalofjavascript.com/getting-started-mqtt/ (dernière consultation le 1 mai 2018).

[5] Creating an Add-on for the Project Things Gateway | Mozilla Hack. https://hacks.mozilla.org/2018/02/creating-an-add-on-for-the-project-things-gateway/ (dernière consultation le 14 avril 2018).

[6] Getting Started with Node.js and MQTT. https://blog.risingstack.com/getting-started-with-nodejs-and-mqtt/ (dernière consultation le 14 avril 2018).

[7] MQTT Essentials Part 6 - Quality of Service Levels. https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels (dernière consultation le 11 juillet 2018).

[8] Writing OpenAPI (Swagger) Specification Tutorial. https://apihandyman.io/writing-openapi-swagger-specification-tutorial-part-2-the-basics/ (dernière consultation le 17 mars 2018).

[9] Node.js RESTful API. https://www.tutorialspoint.com/nodejs/nodejs_restful_api.htm (dernière consultation le 21 fevrier 2018).

[10] Web API Reference | Spotify for Developers. https://developer.spotify.com/documentation/web-api/reference/ (dernière consultation le 2 juillet 2018).

[11] spotify-web-api-js (master) | A client-side JS wrapper for the Spotify Web API. https://doxdox.org/jmperez/spotify-web-api-js (dernière consultation le 2 juillet 2018).

[12] Adding Swagger To Existing Node.js Project | CloudBoost. https://blog.cloudboost.io/adding-swagger-to-existing-node-js-project-92a6624b855b (dernière consultation le 17 mars 2018).

[13] Nordic Thingy-52 v2.1.0 - Firmware architecture. https://nordicsemiconductor.github.io/Nordic-Thingy52-FW/documentation/firmware_architecture.html (dernière consultation le 19 juin 2018).

[14] Walkthrough of Nordic Thingy-52 Node.js Raspberry Pi demos. https://devzone.nordicsemi.com/b/blog/posts/walkthrough-of-nordic-thingy52-nodejs-raspberry-pi (dernière consultation le 30 mars 2018).

[15] Web of Things (WoT) Thing Description. https://www.w3.org/TR/wot-thing-description/ (dernière consultation le 30 juillet 2018). 1, 18

[16] Interact with Bluetooth devices on the Web | Google Developers. https://developers.google.com/web/updates/2015/07/interact-with-ble-devices-on-the-web (dernière consultation le 17 juin 2018).

[17] Web Bluetooth. https://webbluetoothcg.github.io/web-bluetooth/ (dernière consultation le 17 juin 2018).

[18] Web Thing API. https://iot.mozilla.org/wot/ (dernière consultation le 30 juillet 2018). 1, 18

[19] David Wettstein. RESTful services and automation for comfort-oriented smart devices. Master thesis, Department of Informatics, University of Fribourg, Switzerland, December 2017.