Projet FabLab IoT

Développement d'une interface de pilotage de périphérique domotique

Auteurs :

Elie CASTANG

Thibault GUERINEL

Elise MÉTAYER

Ismael SYLLA

Aubry TONNERRE

Mai 2024

Table des matières

Table des figures

1. INTRODUCTION

Ce projet fait suite à un projet initié au semestre 7 dans le cadre de notre formation en ESIR2 spécialité Informatique parcours IoT. Nous avons consacré le semestre 8 à la réalisation de notre projet en essayer de respecter ce qui avait été défini au semestre précédent. Pour rappel, ce projet a pour but de développer l'esprit de synthèse, le travail en équipe, d'apprendre à gérer un projet dans son ensemble (organisation, respect des délais, satisfaction du client) et de développer de nouvelles compétences (méthodologies, langages de programmation, protocoles de communication).

Nous allons travailler sur le développement d'une interface de pilotage de périphériques domotiques. Nous avons choisi de travailler sur ce projet, car il permet un rendu concret grâce à l'interaction des objets du quotidien entre eux. Nous avions envie d'en apprendre plus sur la domotique qui est un domaine dans lequel nous pourrions travailler et qui est amené à perdurer.

La domotique est l'ensemble des techniques de l'électronique, de physique du bâtiment, d'automatisme, de l'informatique et des télécommunications utilisées dans les bâtiments. Elle a pour but d'être plus ou moins « interopérable » et de permettre de centraliser le contrôle des différents systèmes et sous-systèmes de la maison et de l'entreprise.

L'avantage principal de la domotique est sa capacité à faire d'importantes économies d'énergie à l'intérieur d'une maison. Si elle est bien conçue, une installation domotique permet de réaliser entre 25 et 30% d'économie énergétique par an. Elle vise à apporter de simples solutions intuitives aux problématiques d'économie d'énergie, de sécurité et de confort en centralisant le contrôle des différents équipements d'une maison ou d'un bâtiment.

Nous allons commencer par représenter le projet et rappeler le cahier des charges. Nous allons ensuite présenter l'ensemble des fonctionnalités en détails et les méthodes et outils qui ont été utilisés pour implémenter la solution. Enfin, nous dresserons un bilan sur le projet ainsi que des pistes d'améliorations aussi bien sur le plan technique qu'en termes de gestions de projet.

2. RAPPEL DE PROJET

a. Présentation du projet

Notre projet se présente sous la forme d'une application web destinée à la gestion des modules domotiques par l'utilisateur. Nous avons mis l'accent sur la conception d'une interface intuitive et accessible, facilitant ainsi la manipulation et l'interaction avec les différents modules domotiques connectés. Nous nous sommes particulièrement concentrés sur l'utilisation exclusive de modules compatibles avec le protocole ZigBee. De plus, nous avons intégré la fonctionnalité permettant à l'utilisateur d'automatiser des actions grâce à des scénarios préétablis.

b. Cahier des charges

Afin de rappeler notre cahier des charges, voici le diagramme pieuvre extrait du premier rapport.

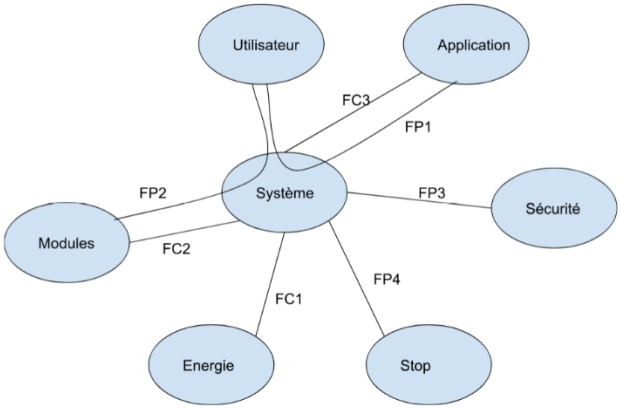

FIGURE 1 - DIAGRAMME PIEUVRE DU PROJET

FP1 : Créer/activer des scénarios

FP2 : Contrôler les modules à distance (mais chez soi)

FP3 : Sécuriser la connexion (ne pas laisser à n'importe qui accéder aux actionneurs)

FP4 : Arrêt manuel d'urgence

FC1 : Être autonome (énergie)

FC2 : Ajout des différents modules sur l'appli

FC3 : Accessibilité (facile à utiliser)

Il est à noter que nous n'avons pas intégré de bouton d'arrêt d'urgence dans notre conception. Cependant, nous avons implémenté un système de gestion des privilèges, où certaines actions sont protégées par un mot de passe.

c. Structure générale

Notre projet est divisé en trois grandes parties, le Front avec principalement l'interface homme/machine, le Back avec le serveur et la base de données ainsi que la routine des scénarios. Le serveur est composé d'une API Express et d'un analyseur de log sur Zigbee2Mqtt. Pour visualiser l'ensemble voici un schéma de la structure de notre projet :

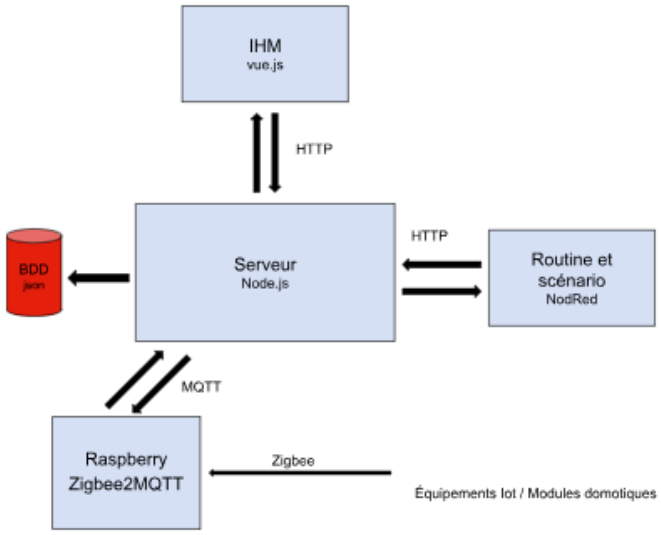

FIGURE 2 – STRUCTURE DU PROJET

Notre structure générale n'a pas subi de modifications majeures par rapport au semestre précédent. Cependant, nous avons introduit un nouveau service pour la gestion des routines dans notre projet. Cette composante a été développée sous Node-RED et communique avec le serveur via des requêtes HTTP, similairement à l'interface homme-machine (IHM).

3. FONCTIONNALITES

L'objectif de cette section est de fournir une description factuelle de toutes les fonctionnalités développées dans chaque partie du projet.

a. Automatisation

Cette partie a pour vocation de permettre à l'utilisateur d'automatiser des actions dans sa maison et de lui permettre de créer des actions en s'intégrant à une multitude de services existants.

C'est afin de répondre à cette problématique que nous avons fait le choix d'utiliser NodeRed qui est un outil de développement low-code basé sur les flux pour la programmation visuelle. Ceci favorisant ainsi l'accessibilité de notre solution pour des utilisateurs n'ayant pas de connaissances en langage de programmation. Une autre raison de ce choix est la possibilité pour l'utilisateur d'installer sur l'interface NodeRed des plugins (ou palettes) développés par la communauté. Ceci permet ainsi un large éventail d'automatisations réalisables pour l'utilisateur.

FIGURE 3 – LOGO NODE-RED

Intégration du serveur NodeRed dans notre solution domotique

Notre idée initiale a été de mettre le serveur NodeRed sur une machine virtuelle de l'ISTIC permettant ainsi à l'utilisateur d'y accéder à distance et nous assurant une connectivité à internet. En définitive, nous avons fait le choix de mettre le serveur NodeRed sur la RaspberryPi du projet. Ceci pour deux raisons : des problèmes de pare-feu avec les différents réseaux de l'ISTIC et la volonté de revenir à une dimension "all-in-one" de la solution.

Le serveur est donc installé sur la RaspberryPi directement sur l'OS. Afin de simplifier au maximum l'expérience utilisateur, nous avons intégré via un Iframe l'interface NodeRed dans un onglet de l'interface Web que nous avons développé.

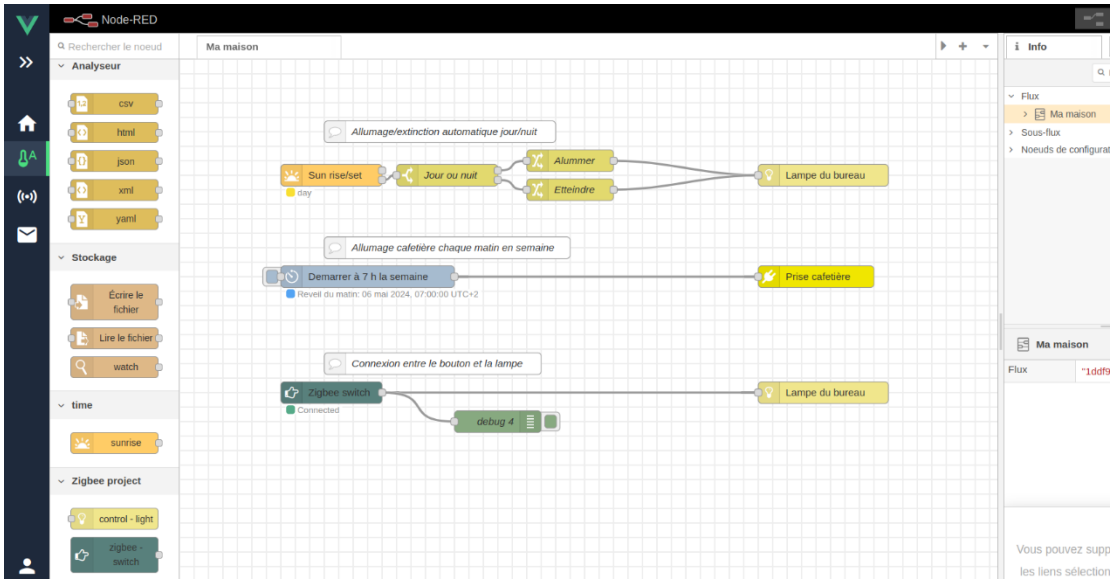

FIGURE 4 – INTERFACE NODE-RED

Création d'une palette NodeRed

L'objectif étant que l'utilisateur puisse, de manière simple, créer des automatisations avec NodeRed. La meilleure manière est de créer des nœuds NodeRed pour chaque type d'objet que nous avons. Une palette est un ensemble de nœud qui peut être mise à disposition de la communauté NodeRed et s'installer en quelques clics.

Côté technique, ces nœuds utilisent les API qui ont été développés pour interagir avec l'interface utilisateur que nous avons développées. Nous utilisons les requetés suivants :

- GetDetails

La fonction getDetails renvoie le détail d'un appareil. Voici un exemple de ce que nous renvoyons.

{time: '2024-05-02 15:28:39:', battery: 39.5, humidity: 59.74, linkquality: 132, temperature: 19.95, voltage: 2800, deviceId: '0x00124b00226717d1', deviceStatus: 'connected', return: 'Ok'}

Nous renvoyons la dernière valeur de chaque paramètre avec des requêtes POST aux URL suivantes :

- /action/plug/id
- /action/light/id

Les paramètres suivants sont attendus :

- L'ID de l'appareil à actionner dans l'URL
- Le type d'action à effectuer en paramètre de la requête {"action":"ON"}

Ces deux API sont très similaire dans leur fonctionnement, elles récupèrent les paramètres de la requête POST puis vérifient si un paramètre d'action correct (ON, OFF, TOGGLE) a été passé. Une fois cette étape effectuée la fonction envoie un message MQTT sur le topic correspondant à notre configuration Zigbee2MQTT afin de transmettre l'action à l'objet.

Le nœud de configuration du serveur d'API

Bien que nous ayons fait le choix d'une solution tout en un, nous avons voulu laisser la possibilité de configurer manuellement l'adresse du serveur au cas où un utilisateur averti choisirait d'utiliser la palette sur un autre serveur NodeRed.

Dans ce nœud l'utilisateur est invité à choisir le port et l'adresse du serveur d'API de notre solution.

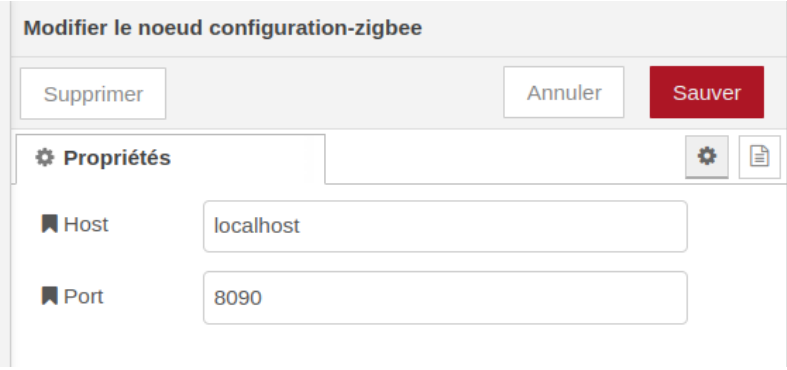

FIGURE 5 - ECRAN DE REGLAGES DU NŒUD DE CONFIGURATION DU SERVEUR D'API

Les actionneurs

Nous avons développé deux nœuds correspondant à nos objets de type : actionneur.

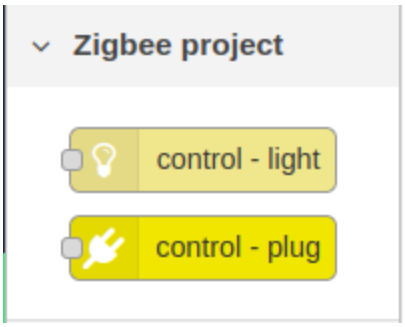

FIGURE 6 - EXTRAIT DE LA PALETTE NODERED DEVELOPPEE

C'est ainsi que nous avons un nœud pour la lampe et la prise de notre projet. Chaque nœud se paramètre par l'utilisateur en entrant l'ID de l'objet qu'il souhaite contrôler.

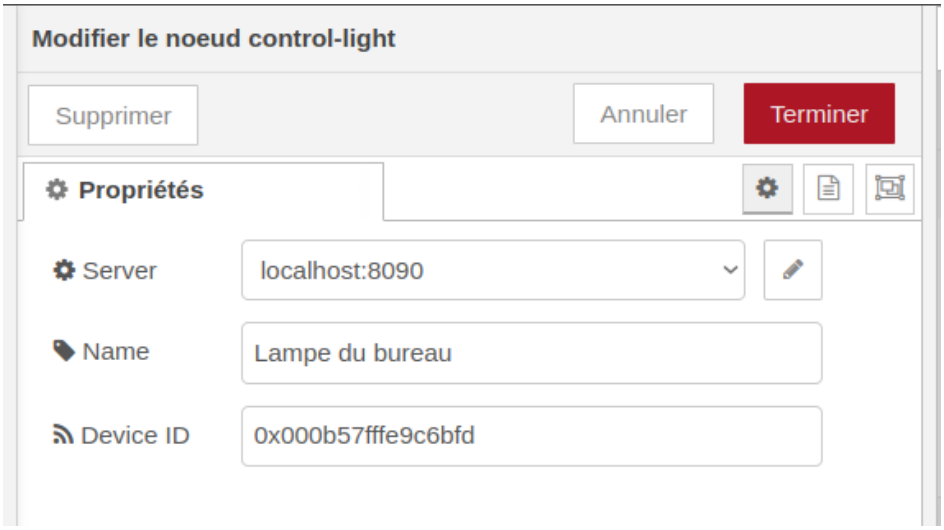

FIGURE 7 - ECRAN DE REGLAGE DU NŒUD CONTROL-LIGHT

Une fois la configuration effectué l'utilisateur peut envoyer dans le payload les commandes suivantes : ON, OFF et TOGGLE.

Les capteurs

Nous avons développé un nœud correspondant à l'interrupteur multi boutons que nous avons. Ce nœud se paramètre par l'utilisateur en entrant l'ID de l'objet qu'il souhaite contrôler.

Une fois la configuration effectué l'utilisateur peut récupérer dans le payload le nom des boutons sur lequel l'utilisateur appuie.

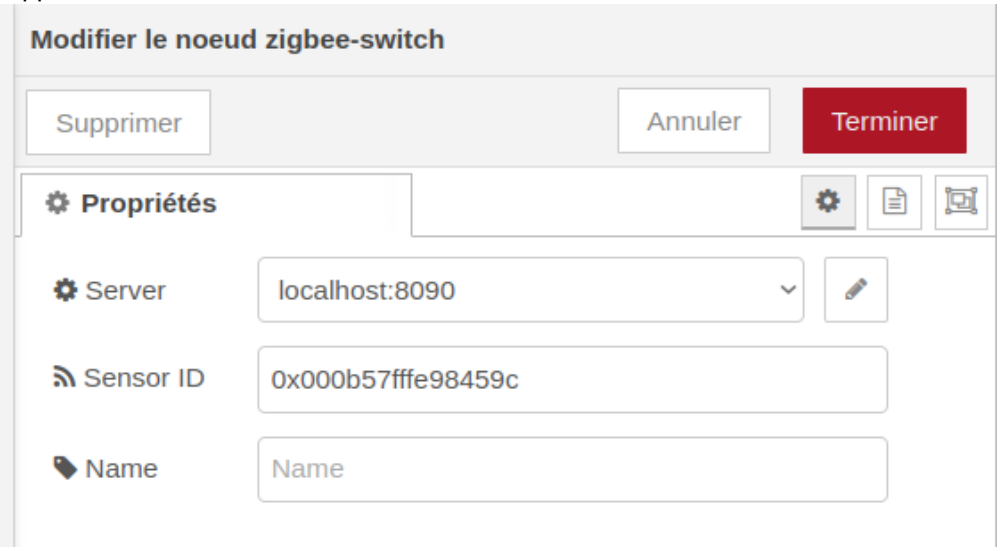

FIGURE 8 - ECRAN DE REGLAGE DU NŒUD ZIGBEE-SWITCH

> Note : le temps nous ayant fait défaut, nous n'avons pas créée tous les capteurs disponibles.

Exemples d'automatisation

Avec les nœuds de la palette uniquement

Imaginons le cas suivant : "Notre utilisateur souhaite connecter un interrupteur à la lampe de son bureau". Il n'a alors qu'à relier le noeud du bouton à celui de la lampe.

FIGURE 9 - EXEMPLE D'AUTOMATISATION N°1

Avec des palettes développées par la communauté

Imaginons le cas suivant : "Notre utilisateur souhaite que son café soit prêt chaque matin quand il se lève pour partir au travail". Il faut alors télécharger une palette de la communauté permettant de gérer les heures et jours de la semaine puis de connecter le nœud téléchargé à la prise de la cafetière.

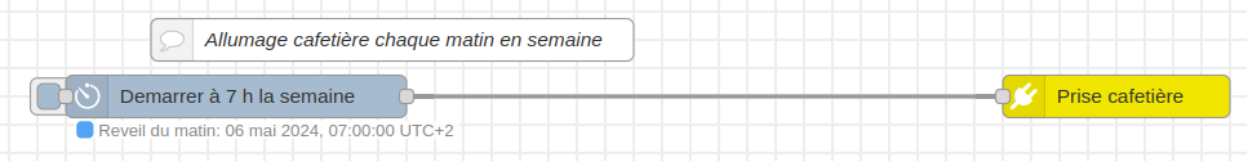

FIGURE 10 - EXEMPLE D'AUTOMATISATION N°2

b. Fonctionnement du back

Dans notre approche, nous avons élaboré un système pour examiner de près les activités de notre Zigbee2mqtt. Cela a nécessité la création d'un outil sur mesure, notre script de surveillance des logs, qui joue un rôle crucial dans notre processus. Ce script fonctionne en permanence en arrière-plan, scrutant attentivement un fichier de logs spécifique où sont consignées toutes les actions essentielles de notre système. À chaque changement détecté dans ces logs, tel qu'une nouvelle connexion de capteur ou l'émission de nouvelles données, notre script est instantanément alerté et entre en action. Il extrait alors minutieusement les détails de cet événement fraîchement enregistré, y compris le type d'action effectuée et les données associées. Ensuite, ces informations sont soumises à un processus d'analyse approfondi, visant à démystifier le sens et l'impact de chaque action. Une fois cette analyse achevée, les résultats sont soigneusement intégrés à notre base de données, un réservoir structuré au format JSON, où ils trouvent leur place parmi d'autres données vitales. Cette base de données devient ainsi notre référentiel central, permettant de retracer l'historique de toutes les activités de nos capteurs et de notre système Zigbee2mqtt. En synthèse, cette approche nous offre un aperçu précieux sur le fonctionnement interne de notre système, nous permettant de prendre des décisions éclairées et de mettre en œuvre des améliorations continues pour une performance optimale.

c. Tableau de bord

Cette partie sera consacrée au tableau de bord, c'est-à-dire l'onglet 'Home' de l'application. Sur ce tableau de bord, 100% personnalisable selon le bon vouloir de l'utilisateur, seront affichés des widgets. Ce tableau de bord sert à montrer à l'utilisateur les états des capteurs qu'il souhaite voir. Chaque module renvoie des informations et elles sont ensuite traitées avant d'être affichées sous forme de widget pour une meilleure lecture et interprétation de celles-ci. Son mode de fonctionnement est simple d'utilisation puisque les icones représentent chaque fonction.

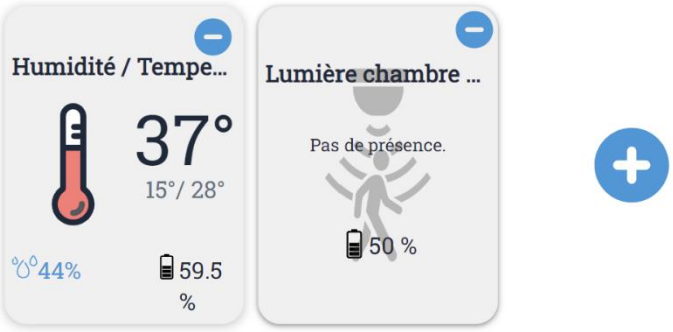

FIGURE 11 – APERÇU DU TABLEAU DE BOARD

Par exemple, comme vous pouvez le constater sur la figure ci-dessous l'icône + sert à ajouter des widgets. Et chacun d'eux possède une icône – qui sert à les supprimer du tableau de bord.

Les widgets ont des templates généraux et c'est l'utilisateur qui choisit ensuite à quel module domotique il sera relié. Pour ce faire il doit cliquer sur le texte 'choisir un module' qui est présent sur tous les widgets sans module, ce texte affichera une liste de module correspondant au template du widget. Par exemple les widgets avec des templates de type température ne peuvent être reliés qu'a un module qui renvoie une température. Sur la figure si dessous nous pouvons voir un widget sans module relié, un widget en cours de choix pour être relié à un module

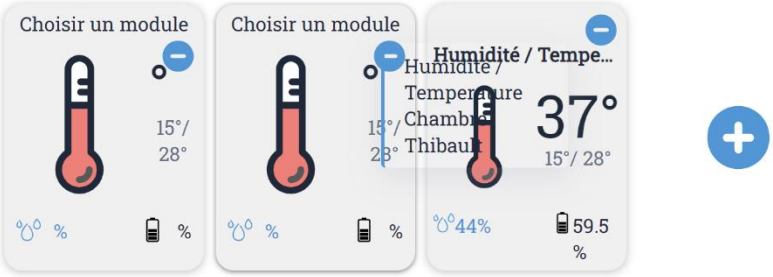

FIGURE 12 – APERÇU DES WIDGETS DE TEMPERATURE

Une fois qu'un widget est associé à un module nous appelons l'API avec le nom de celui-ci pour y récupérer toutes les informations utiles, ci-dessus nous avons la température actuelle, l'humidité et l'état de la batterie du module.

L'API reçoit donc une requête de ce type :

Description	Type requête	Url	Paramètre	Réponse

Récupération des data	GET	/ListTypeDevice	Type
			Id
			Name
				"type": {
				"id": {
				"deviceId": "id",
				"deviceName": "name",
				"deviceStatus": "ON",
				"info": {
				"time": "2024-03-18 11:25:00",
				"battery": 50,
				"linkquality": 171,
				"occupancy": false
				}
				}
				}

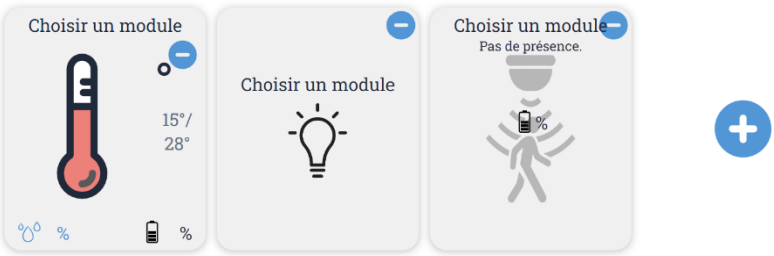

FIGURE 13 – APERÇU DES WIDGETS DISPONIBLES

Voici donc tous les templates de widgets pouvant être associés à des modules domotique que l'utilisateur a chez lui.

Dans cette partie, la réalisation prend un certain temps pour décider du design, faire la connexion à l'API et rendre le tout dynamique. Cependant l'étape qui a été la plus difficile est celle de la mise en store des widgets. En effet il serait contraignant pour l'utilisateur d'avoir à refaire entièrement son tableau de bord à chaque connexion. Nous avons donc choisi d'enregistrer les informations utiles dans le local Storage. Ainsi pour chaque ajout ou retrait d'un module les informations sur l'état de la page son sauvegarder et réutilisées en cas de refresh de celle-ci.

widgets	[{"name":"temperature","module":"Humidité / Temperature Chambre Thibault"},{"name":"switch"},{"name":"presence"}]

FIGURE 14 – LOCAL STORAGE

Nous avons donc le type de widget et s'il a été défini, son module associé.

Back

L'objectif de la requête '/listDeviceType' est de récupérer une liste de types d'appareils à partir d'un fichier JSON et renvoyer cette liste au client qui a fait la requête.

Nous commençons par lire le fichier 'table_Module.json' du répertoire 'BDD'. Puis, sur chaque appareil en base de données on stocke dans `element` l'identifiant du module, le nom du module, le statut du module et un objet vide `info`.

Après avoir parcouru toutes les clés du module, il parcourt chaque type dans le tableau `type`. Si le type n'est pas l'un des types exclus ('nom', 'init', 'time', 'linkquality', 'voltage', 'battery_low', 'tamper', 'battery'), il ajoute le type à l'objet `result` comme une nouvelle propriété si elle n'existe pas déjà, et ajoute l'objet `element` à cette propriété si le module contient ce type.

Une fois tous les modules parcourus, il renvoie l'objet `result` comme réponse JSON à la requête GET.

En résumé, nous organisons les modules par type et renvoyions un objet où chaque clé est un type de module et chaque valeur est un objet contenant les informations des modules de ce type.

Voici un exemple de renvoie :

```
humidity: {
  '0x00124b00226717d1': {
    deviceId: '0x00124b00226717d1',
    deviceName: '',
    deviceStatus: 'ON',
    info: [Object]
  }
},
temperature: {
  '0x00124b00226717d1': {
    deviceId: '0x00124b00226717d1',
    deviceName: '',
    deviceStatus: 'ON',
    info: [Object]
  }
},
```


FIGURE 15 - EXEMPLE DE RENVOIE DE LA REQUETE LISTDEVICETYPE

On observe que le module ‘0x00124b00226717d1’ est un capteur de température et d’humidité. Nous n’avons, dans la base de données, que ce module qui réponde à cette requête.

d. Connexion et gestion de module

Nous allons dans cette partie d’écrire la fonctionnalité qui concerne la gestion des périphériques ZigBee. Nous allons décrire comment ajouter de nouveau périphérique à l’interface, comment les édités, comment les supprimés et comment avoir un historique.

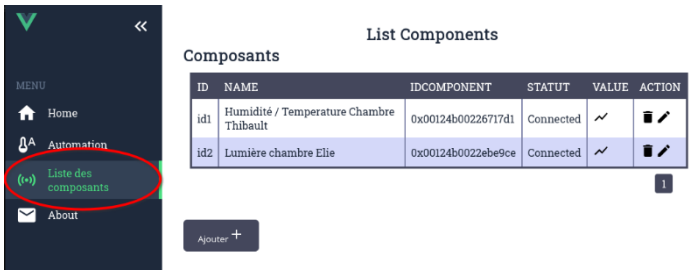

FIGURE 16 – PAGE DE GESTION DE LA LISTE DES COMPOSANTS

L’ensemble des fonctionnalités décrite dans la partie suivante se trouvent sur l’onglet ”Liste des composants” de l’application.

Ajout du périphérique

Front

L'ajout de périphériques se fait par l'intermédiaire du bouton "Ajouter" situer en dessous du tableau des composants. Le scénario normal consiste à cliquer sur le bouton "Ajouter", ce qui envoie une requête

au serveur. En retour, le serveur renvoie l'ensemble des capteurs disponibles pour l'ajout. Une fois choisi, nous modifions le nom du capteur dans un formulaire, puis celui-ci est ajouté à la base de données.

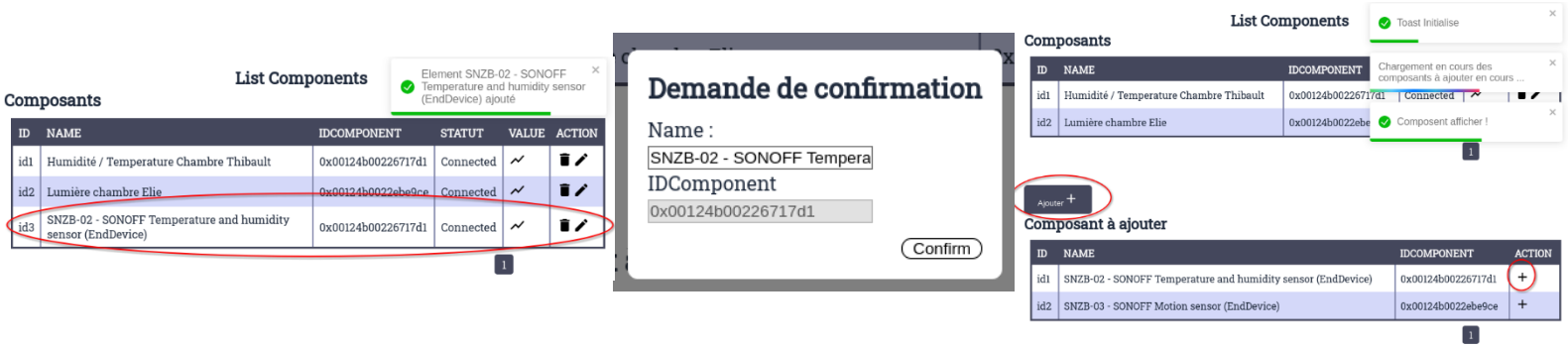

FIGURE 17 – ETAPE DE REALISATION POUR L'AJOUT D'UN COMPOSANT

Ceci est le scénario normal. Cependant, suite à un problème de liaison avec le serveur zigbee2mqtt, nous avons été contraints de simplifier la procédure d'ajout. Désormais, lorsque l'utilisateur clique sur le bouton "Ajouter", les nouveaux composants sont automatiquement ajoutés à la liste des composants du système. Il revient alors à l'utilisateur de décider s'il souhaite les retirer ou non.

Back

Afin de gérer l'ajout de module, nous envoyons une requête au server Zigbee2MQTT pour qu'il ouvre ses ports de connexions. Ils détectent alors autour de lui les appareils disponibles et les ajoutes automatiquement. Nous récupérons dans les logs les informations de connexions et nous les ajoutons en base de données. Le front nous envoie une requête pour afficher les informations que nous avons en base de données.

Edition du périphérique

Front

Lors de l'édition d'un périphérique, l'utilisateur à la possibilité de cliquer sur un bouton qui lui permettra d'accéder à un formulaire lui permettant de modifier le contenu du tableau. Une fois le nouveau nom choisi par l'utilisateur, une requête HTTP est transmise à l'API Rest qui sera charger de faire la modification adaptée. À noter que si l'utilisateur ne rentre pas de nom dans le fichier le forme ne s'enverra pas et si le nom transmis contient des '/' le système ne fonctionnera pas.

Description	Type requête	Url	Paramètre	Réponse

Modifier nom	POST	/writeName/:name/:id	id (string) : id du device
			name (string) : nom du Device
				JSON {
				"deviceId" :
				"deviceName" :
				"deviceStatus" : update
				"Ok"
				}

Nous observons ici les étapes qui s'enchaine lors de la modification.

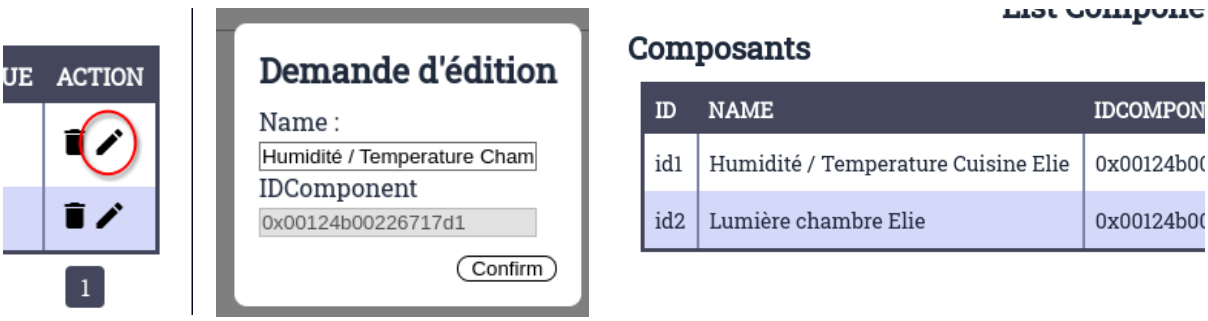

FIGURE 18 – ETAPE DE REALISATION POUR L'EDITION D'UN COMPOSANT

Back

Nous modifions simplement le fichier JSON avec les informations reçues. Nous recherchons l'id du module dans la base et modifions le nom avec celui envoyé dans la requête.

Suppression du périphérique

Front

Lors de la suppression d'un périphérique, l'utilisateur à la possibilité de cliquer sur un bouton qui lui permettra d'accéder à un formulaire lui demandant confirmation pour supprimer le contenu du tableau sélectionné. Une fois le nouveau la confirmation faite, l'interface graphique contacte l'api REST pour la suppression du périphérique. Une fois la suppression faites, l'IHM affiche comme quoi l'opération est réussite et actualise la table des périphériques.

Description	Type requête	Url	Paramètre	Réponse
Suppression composant	DELETE	/delete/:id	Id (string) : id unique device	JSON { "OK" }

Nous observons ici les étapes qui s'enchaine lors de la suppression.

FIGURE 19 - FIGURE 20 – ETAPE DE REALISATION POUR LA SUPPRESSION D'UN COMPOSANT

Back

Nous supprimons le module de la base de données et nous envoyons une requête à Zigbee2MQTT pour qu'il le supprime de son côté aussi afin d'être toujours à jour. La requête envoyée contiendra les informations suivantes :

```
var message = `{"id":"${id}","force":true}`;
var topic = "zigbee2mqtt/bridge/request/device/remove";
```

Récupération de l'historique d'un périphérique

Front

Lors de la consultation de l'historique pour un périphérique choisi, l'utilisateur a la possibilité de cliquer sur un bouton qui lui permettra d'accéder à un modal lui fournissant l'ensemble des données fourni par le capteur sous la forme de graphique. En fonction du type de données traité par les capteurs, l'affichage se personnalise et propose N tables en fonction des grandeurs physiques détecté et mesuré.

Description	Type requête	Url	Paramètre	Réponse
Historique sur les 30 dernières valeurs pour un appareil donné.	GET	/device/:id/data	id (string) : id du device	JSON { "unit" : [unit1, unit2...] "name" : [nom1, nom2...] "valueGraph" : valeurFormatéAffichag }

Nous observons ici les étapes qui s'enchaine lors de la consultation de l'historique de valeurs.

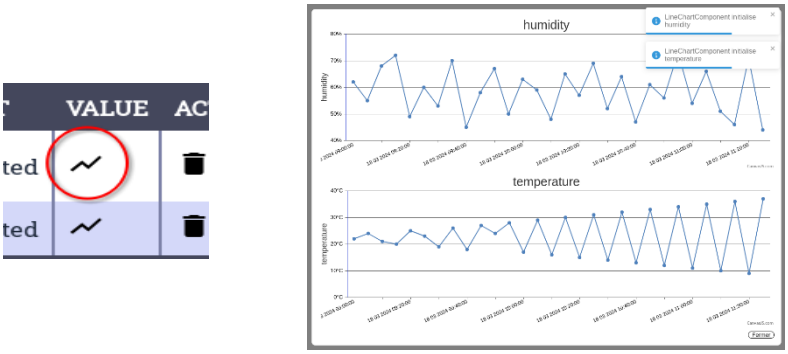

FIGURE 21 – ETAPE POUR L'AFFICHAGE DE L'HISTORIQUE DES MESURES D'UN COMPOSANTS

Back

Grâce à notre analyse de Log nous arrivons à récupérer les données envoyées par le capteur. Voici un exemple de données brutes que nous récupérons :

```
info 2023-10-23 15:20:25: MQTT publish: topic 'zigbee2mqtt/0x00124b00226717d1', payload '{"battery":59.5,"humidity":62.34,"linkquality":171,"temperature":20.93,"voltage":2900}'
```

Nous obtenons un résultat de ce type :

FIGURE 22 - EXEMPLE D'UN APPAREIL DANS NOTRE BASE DE DONNEE

Nous traitons chaque variable et nous remplissons notre table de données.

Quand le Front demande les informations elles sont toutes disponibles dans notre base de données. Dans le cas où il serait demandé de fournir 30 valeurs, nous ajoutons des 0 au début de la liste.

e. Privilèges utilisateur

Dans le développement de notre application, nous avons intégré un système de gestion des utilisateurs qui revêt une grande importance. Ce système est conçu pour superviser les actions entreprises par les utilisateurs au sein de notre plateforme. Une caractéristique clé de ce système est la gestion des autorisations. En effet, certaines fonctionnalités de notre application, telles que la suppression d'un appareil du tableau de bord, nécessitent un niveau d'autorisation spécifique pour être exécutées. Ainsi, avant de permettre l'exécution de toute action sensible, notre application vérifie toujours si l'utilisateur possède les autorisations requises. Cette approche garantit que seuls les utilisateurs autorisés peuvent entreprendre des actions qui pourraient avoir un impact sur les données ou les paramètres de l'application. Par exemple, un utilisateur régulier pourrait ne pas avoir la permission de supprimer un appareil critique, ce qui évite les erreurs accidentelles ou les dommages potentiels. En intégrant cette gestion d'utilisateur, nous renforçons la sécurité de notre application en limitant l'accès aux fonctionnalités sensibles en fonction des droits et des responsabilités de chaque utilisateur. De plus, cela contribue à personnaliser l'expérience utilisateur en offrant un environnement adapté aux besoins et aux autorisations spécifiques de chaque individu. En fin de compte, cette approche aide à instaurer un climat de confiance et de fiabilité, ce qui est essentiel pour assurer le succès et la satisfaction des utilisateurs.

4. COMMENTAIRE SUR L'ENVIRONNEMENT DE DEVELOPPEMENT

Le serveur de notre environnement de production est une RaspberryPi que nous avions besoin d'avoir physiquement à nos côtés pour qu'elle puisse interagir avec les objets via le protocole ZigBee. La difficulté pour nous, a donc été de pouvoir développer notre code puis de l'envoyer à notre serveur pour tester.

Le choix d'utiliser le réseau WIFI-IOT-V mis à disposition par l'ISTIC nous a semblé être la solution la plus simple puisque l'authentification au réseau se fait sur la déclaration d'une adresse MAC auprès de la DSI. Hélas, nous nous sommes heurtés à plusieurs difficultés avec ce réseau en raison de règles de pare-feu strictes, même pour des échanges avec les autres réseaux de l'Université. Ainisi, nous avons contacté la DSI de l'ISTIC pour solliciter leur aide et demander des modifications dans leurs configurations afin de nous faciliter le développement. La DSI de l'ISTIC a, par exemple, modifié la zone DNS de esir.univ-rennes.fr pour nous créer un nom de domaine pour le projet et ajouter une réservation d'adresse IP dans leur DHCP pour que notre serveur soit toujours joignable à la même adresse.

Nous avons aussi échangé avec eux pour des blocages du protocole NTP ou des messages ICMP.

Nous souhaitons suggérer pour un prochain projet d'ajouter au matériel à disposition un routeur WiFi qui permettrait d'avoir une gestion totale du réseau et de mieux travailler ainsi.

5. PISTES D'AMELIORATIONS

Si nous prenons un peu de recul sur le développement de notre API Express, nous aurions dû scinder une partie de l'API en sous-service, afin de répartir l'intelligence. Les requêtes mériteraient une meilleure nomenclature de nommage afin de faciliter son utilisation et une meilleure compréhension.

Afin d'avoir une meilleure performance de notre analyse log, il vaudrait mieux s'abonner aux Topics pour avoir les informations en direct et traiter avec plus de précision les erreurs de connexion.

En ce qui concerne le front-end, nous avons constaté que la gestion de la communication avec l'API Rest pourrait être centralisée dans un sous-service, facilitant ainsi les interactions. De plus, pour la table de gestion des périphériques, nous avons adopté un certain formalisme en utilisant des icônes, par exemple pour accéder à l'historique. Cependant, cette disposition peut ne pas sembler intuitive pour tous les utilisateurs. Nous pourrions envisager d'ajouter des paramètres pour afficher plus de détails ou pour personnaliser davantage notre application.

6. CONCLUSION

Pour conclure, l'objectif de ce projet était de développer une application de gestion de périphériques domotiques. Tout au long de ce projet, nous avons défini des objectifs de développement à l'aide d'un cahier des charges.

Le bilan est que, dans l'ensemble, la plupart des fonctionnalités ont été développées. De manière globale, nous avons pu mettre en application des compétences acquises dans d'autres disciplines ou encore en développant de nouvelles compétences, notamment la création d'une interface graphique pour l'utilisateur, la construction de nœuds sur Node-RED, et les processus d'automatisation afférents.

Après avoir pris du recul, nous avons compris que certaines de nos implémentations n'étaient pas parfaites, notamment la partie dite du "backend" ainsi que le "frontend".

Cependant, nous sommes globalement satisfaits du travail produit et de ce que nous pouvons présenter de fonctionnel à la démonstration de fin de projet.

Ainsi, pour la suite de notre cursus, ce projet correspond à nos objectifs pour le futur. Il nous a également permis de travailler au sein d'une équipe où chacun apporte ses forces et ses faiblesses dans les différents domaines techniques abordés, un contexte que nous retrouverons en entreprise dans quelques mois.

Table des Annexes

Documentation API - ESIR2 S8 Fablab					
Description	**type requête**	**url**	**paramètre**	**réponse**	**Privileges**
modifier nom	POST	/writeName/:name/:id	id (int) : id du device name (int) : nom du Device	JSON { “deviceId” : “deviceName” : “deviceStatus” : update “Ok”}	✕
lister les devices avec toutes les params et la dernière valeures	GET	/listDevice URGENT		JSON { “deviceId” : “deviceName” : “deviceStatus” : “listInfo” : param1 : value1 param2 : value2] }	✕
lister les devices avec toutes les params et la dernière valeure MAIS les devices trié par type	GET	/listDeviceType		"JSON temperature{ {“deviceId” : “deviceName” : “deviceStatus” : “listInfo” : param1 : value1 param2 : value2}, {“deviceId” : “deviceName” : “deviceStatus” : “listInfo” : param1 : value1 param2 : value2}, }, occupancy { ... }"	✕

lister les devices avec les	GET	/listDevice/details URGENT		JSON { “deviceId” : “deviceName” : “deviceStatus” : “listInfo” : param1 : val1 val2… param2 : val1 val2… }	✕
rechercher des devices (par nom). Renvoie le ou les noms de device qui correspond à la distance de hamming la plus petite parmis la liste des noms de device	GET	/search/:str	name (string) : nom du device	JSON { “module find" : name}	✕
rechercher des devices (par nom). Renvoie le ou les noms de device qui correspond au première lettre lié à la recherche : toto1 toto2 tata str : to réponse : toto1 / toto2	GET	/searchList/:str	name (string) : nom du device	JSON { name1, name2, nam3, ... }	✕

obtenir l'état de 1 device
 	
FAIT
	GET	/getState/:id	id (int) : id du device	JSON {
				"deviceId" : XXX
				"deviceStatus" : connected or no connected
				"Ok"
				}
					✕

obtenir info device	GET	/getDetails/:id	id (int) : id du device	JSON {
				"deviceId" : XXX
				"deviceName" :
				"listInfo" :
				param1 : value1
				
				param2 : value2
				
				"deviceStatus" : connected or no connected
				"Ok"
				}
					✕

obtenir info device
 	
Historique sur les 30 dernières valeurs maximum. Sans "battery", "linkquality", "voltage"
 	
pour un id donnée
	GET	/device/:id/data	id (int) : id du device
			
			unit : °C/%
				JSON {
				"unit" : [unit1, unit2...]
				"name" : [nom1, nom2...]
				"valueGraph" : valeurFormatéAffichage
				}	

vérifier si le mot de passe est correct	GET	/askPermission/:password		JSON{true ou false}	✕

changer le mot de passe	POST	/changePassword/:password		JSON { "OK" }	
delete composant	DELETE	/delete/:id		JSON { "OK" }	
Allumer/eteindre prise	POST	/action/plug/:id	id (int) : id du device | dans le corps JSON {"object" : '[id_de_lobjet]', "action":"ON|OFF"|TOGGLE}	JSON { "OK" }	
Lancer procédure connexion	GET	/allowAddDevice	dans le corps JSON {"object" : '[id_de_lobjet]', "action":"ON|OFF"}	JSON { "OK" }	
				JSON{ [{id = id1, name = name1} , {id = id2, name = name2}] }	

ajout device, j'enrigistre ID, j'enregistre le nom	GET	/addModule/:id/:name		JSON { return : “Ok” "NON OK" }	✕

ANNEXE 1 - TABLE REQUETE API