

Année académique 2018-2019

HAUTE ECOLE DE LA COMMUNAUTE FRANCAISE EN HAINAUT

Catégorie technique

8A Avenue Victor Maistriau – 7000 Mons

**Développement de kits domotiques prémontés**

Travail de fin d'études réalisé en vue de l'obtention du titre de bachelier en Informatique et systèmes, orientation réseaux et télécommunications

Promoteur : Lerat Jean-Sébastien Étudiant(s) : Veys Florent

Hempte Maxime

Remerciements

Tables des matières

[Tables des figures – Listes des tableaux 2](#_Toc10642443)

[Tables des sigles 3](#_Toc10642444)

[Abstract étendu 4](#_Toc10642445)

[Introduction 5](#_Toc10642446)

[Choix de la Distribution 6](#_Toc10642447)

[Image du docker 6](#_Toc10642448)

[Raspberry 6](#_Toc10642449)

[Objectifs 6](#_Toc10642450)

[Communication 7](#_Toc10642451)

[Protocole de communication 7](#_Toc10642452)

[MQTT 8](#_Toc10642453)

[Mosquitto 9](#_Toc10642454)

[Paho-MQTT 10](#_Toc10642455)

[Support de communication 10](#_Toc10642456)

[Gestion des utilisateurs 12](#_Toc10642457)

[La base de données 12](#_Toc10642458)

[Docker et MySql 12](#_Toc10642459)

[Dossier avec les scripts sql 12](#_Toc10642460)

[Installation et configuration MySql 13](#_Toc10642461)

[Construction de l’image et lancer le container comme démon. 13](#_Toc10642462)

[Utiliser la base de données en python 14](#_Toc10642463)

[Flask 16](#_Toc10642464)

[Création de l’API 16](#_Toc10642465)

[Flask-RESTful 17](#_Toc10642466)

[Création de la ressource utilisateur 17](#_Toc10642467)

[Login 18](#_Toc10642468)

[CRUD 20](#_Toc10642469)

[GET 20](#_Toc10642470)

[POST 21](#_Toc10642471)

[PUT 22](#_Toc10642472)

[DELETE 22](#_Toc10642473)

[Conclusion 23](#_Toc10642474)

[Bibliographie 24](#_Toc10642475)

[Lexique 25](#_Toc10642476)

Tables des figures – Listes des tableaux

Tables des sigles

MQTT = Message Queuing Telemetry Transport

IoT = Internet of Things

QoS = Quality of Service

AP = Access Point

Abstract étendu

Introduction

# Choix de la Distribution

## Image du docker

Debian étant beaucoup utiliser et par conséquent ayant une large une communauté, il est facile de trouver des informations sur les services ou des solutions à nos problèmes.

Il existe d’autre distribution linux qui se base sur Debian comme par exemple Ubuntu. Debian s’adressant à un public un peu plus initié, il est plus simple de configurer ces distributions si on sait configurer un Debian.

Un autre avantage de Debian est qu’il est totalement libre. Par conséquent lorsqu’une faille est détectée la communauté peut rapidement trouver une solution.  
Plusieurs architecture est supporté par Debian comme AMD, i386, MIPS, etc.

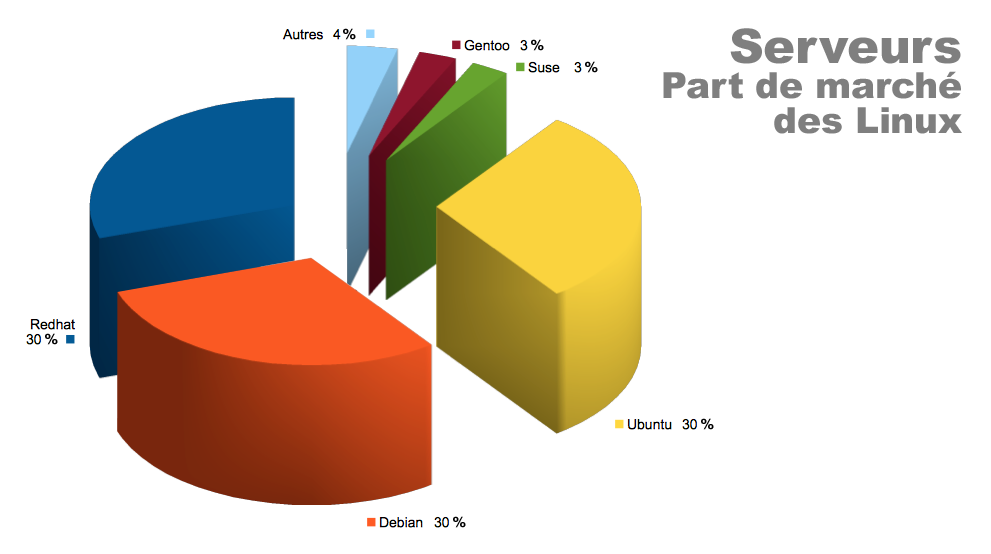


Figure  : Part de marché des Linux

## Raspberry

La distribution que nous avons choisie pour mettre sur nos Raspberry, est bien entendu « Raspbian ». Raspbian est un OS libre et gratuit basé sur Debian, il est optimisé pour justement fonctionner sur un Raspberry. Nous avons installé la version Lite de cet OS ce qui nous permet d’installer uniquement ce qui nous est nécessaire dans notre projet.

# Objectifs

Au commencement de notre réflexion pour la création de notre sujet de TFE, nous avons dû réfléchir à une série d’objectifs que devront réaliser les modules que nous allions développer. Voici donc la liste des fonctionnalités que nous avons brainstormer sur ce sujet :

* Contrôle de la lumière
  + Allumage et extinction.
  + Réglage de l’intensité.
* Contrôle de la présence
  + Pouvoir effectuer des actions telles que, allumer la lumière, allumer la caméra ou dire bonjour ; si un mouvement est détecté dans la pièce.
* Contrôle de l’eau
  + Contrôle de l’ouverture de tuyaux avec des électrovannes.
* Contrôle ménager
  + Contrôle de la machine à café.
* Contrôle Vocal
  + Donner des ordres par la voix
* Gestion d’agenda
  + Ajouter, supprimer, modifier un évènement
  + Récupérer les évènements à une date précis
* Contrôle par application web
  + Donner des ordres, utiliser les fonctionnalités via une application web

Pour éviter que nous nous éparpillions, dans toutes ces fonctionnalités, nous nous sommes focalisés principalement sur la gestion de la lumière, de la présence et le contrôle par l’application Web.

# Communication

## Protocole de communication

Au tout début de la phase de recherche pour la réalisation de notre projet, nous nous étions arrêtés sur les sockets en python, ceux-ci nous permettaient d’établir une connexion entre nos différents modules. Un socket c’est quoi ? c’est comme un « trou » qui laisse passer des choses, dans notre cas cela était une association au niveau de l’OS entre un programme qui tourne en boucle et le port de la machine qui lui a été attribué.

Durant notre période de stage, Maxime ayant parlé de notre sujet de TFE à son maitre de stage, celui-ci nous a donc conseiller d’utiliser le protocole MQTT, qui serait plus adapté au résultat que nous voulions atteindre. J’ai donc effectué des recherches sur ce protocole, pour comprendre son fonctionnement et s’il serait plus adapté que ce que nous avions.

Je me suis très vite aperçu que ce protocole était en faite très utilisé pour tout ce qui concerne des projets qui touchent à l’IoT.

### MQTT

MQTT permet à des clients de publier et ou de s’abonner à des topics d’informations. L’ensemble des clients communiquent avec un broker qui va jouer le rôle de médiateur. Lorsqu’un appareil publie un message sur un topic, le broker transfère le message à tous les autres clients qui ont souscrit à ce même topic.

Le protocole est bidirectionnel, et veille à ce que les appareils restent connectés. En effet, si un appareil « Edge-of-network » perd sa connectivité avec le broker, tous les clients qui y sont lié en seront avertis grâce à la fonctionnalité « Last Will and Testament ».

Ensuite la légèreté et l’efficacité du protocole, il peut être qualifié de léger dans la mesure où les clients sont petits et qu’il utilise de façon efficace la bande passante du réseau.

Il inclut aussi nativement la notion de QoS, permettant de véhiculer des données conformément à certaine exigence en matière de temps de réponse et d’utilisation de bande passante. MQTT s’appuie sur 3 niveaux de QoS :

* Niveau 0 : l’information n’est transmise qu’une seule fois entre le « Publisher » et le « Broker » puis le « client ».
* Niveau 1 : au moins une fois, possibilité de recevoir plusieurs fois le même message si absence d’un accusé de réception
* Niveau 2 : niveau le plus sûr mais le plus lent, quelque soit le nombre de tentative, les informations ne seront reçues qu’une seule fois par le client.

Au niveau de la sécurité des données qui sont transmissent par les appareils, si certaines fonctionnalités ne sont pas mises en place, le protocole peut être assez vulnérable. L’authentification et la transmission des messages peut être faites sans aucun chiffrage. Heureusement, le protocole possède un 2ème port qui permet de passer sur un canal sécuriser (SSL/TLS)

Comme nous recherchions quelque chose qui nous permettais de laisser n’importe quelles personnes développer ses propres objets et de venir les rajouter à notre système. MQTT étant un standard, cela n’a fait que renforcer notre décision de partir sur ce protocole.

En résumé, MQTT est une excellente solution pour assurer la communication entre appareils et son adoption devrait être de plus en plus large à l’avenir. Dans le cadre d’un logiciel écrit en langage C, il simplifie beaucoup la gestion du réseau, notamment les cas de connexion/reconnexion.

### Mosquitto

Pour pouvoir utiliser le protocole MQTT, nous devions mettre en place un « Broker ». Il existe une vingtaine de broker MQTT différent (RabbitMQ, ActiveMQ, JoramMQ, …) qui ont chacun leurs spécificités, qui sont spécialement fait pour certaine utilisation. Comme par exemple, ActiveMQ et JoramMQ qui permet d’ajouter MQTT à un serveur apache ou pour intégrer MQTT en java. RabbitMQ qui lui est plus orienté entreprise avec son offre commerciale.

Enfin nous voilà sur le choix que nous avons effectué pour notre « broker ». Mosquitto est un « broker » open source très léger, qui implémente MQTT v3.1.0, 3.1.1 et la version 5.0. Il convient pour tous les périphériques, des ordinateurs monocartes basse consommation aux serveurs complets.

Etant plus souvent utilisé dans les projets DIY, c’est le broker qui convenait le mieux à notre vision des choses et à nos besoins.

### Paho-MQTT

Durant un projet pendant notre 2ème année de bachelier, nous avions dû configurer un robot qui avait comme carte mère un Raspberry. Nous avions donc créé des scripts en python pour réaliser les différentes actions qu’il nous était demandé. C’est donc pour cela que j’ai cherché une technologie qui permettait d’implémenter le protocole MQTT avec le langage Python.

J’ai donc trouvé une libraire qui convenait exactement à ce que je recherchais, Paho-MQTT est un client MQTT en Python open-source développer par la fondation Eclipse. Paho est un des meilleurs client MQTT en Python car il peut fonctionner sur n’importe quel périphérique qui supporte le multithreading.

Le client implémente des fonctions que l’on peut facilement utiliser pour :

* Se connecter/déconnecter au broker
* Publier des messages sur un topic
* S’abonner/désabonner à un topic

Il possède aussi des fonctions que nous devons mettre en place pour effectuer des actions à la connexion avec le « broker » ou à la réception d’un message, etc.

Comme il s’agit d’un langage permettant de créer des scripts, cela nous permet donc de lancer ces différents codes au moment que nous le voulons.

## Support de communication

Pour le choix du support de communication pour notre projet, nous avions le choix entre différent mode de fonctionnement (Zigbee, Z-wave, WIFI, LoRaWan, …). Chaque technologie avait leurs avantages et leurs inconvénients.

Notre choix c’est porté sur le Wifi, en effet un module Wifi est directement compris avec les Raspberry comparer à d’autres support comme le Zigbee ou le Z-wave qui nécessite un module supplémentaire pour pouvoir émettre ou recevoir sur leur fréquence. De plus, dans le cadre de notre projet, nous n’avions pas besoins d’avoir une technologie qui porte sur de grande distance, comme souvent la plupart des pièces qui possèderait nos modules aurait normalement accès au Wifi.

Le Wifi a aussi comme avantage d’être « visible ». J’entends par cela que l’utilisateur pour voir la portée maximale que le wifi atteindra à l’aide de son smartphone et pourra donc placer en fonction de ses possibilités, les modules à l’endroit de son choix en s’assurant que le module soit à portée de l’AP.

En ce qui concerne le réseau en lui-même, le module primaire jouera le rôle de l’accès point pour les modules secondaire et pour le client. En effet, n’a pas besoin d’avoir une connexion au Wifi avec pour que le système puisse fonctionner. Pour pouvoir utiliser le Raspberry comme un AP, j’utilise RaspAP.

RaspAP est un logiciel qui permet de créer rapidement un Hotspot Wifi. Si le Raspberry est connecté à internet via un câble Ethernet, RaspAP va s’occuper de faire le bridge entre ces deux réseaux, ce qui permettra d’avoir une connexion internet dans le réseau interne au cas où nous implémenterions une fonction de mise à jour des modules secondaire, ou pour toutes autres fonctionnalités qui serait développé par nous ou une autre personne qui nécessiterait un lien avec Internet.

RaspAP possède une interface graphique qui facilite son administration et la gestion du réseau. Cette interface est responsive et peut être donc consulté facilement sur un smartphone ou une tablette. Elle nous apporte beaucoup d’informations qui peuvent nous être utile lors d’un dépannage ou pour vérifier des potentielles erreurs.

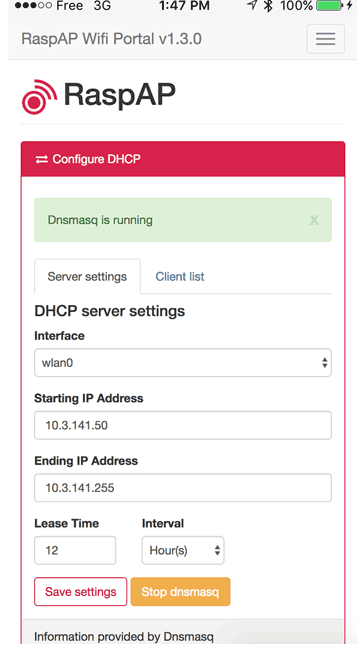
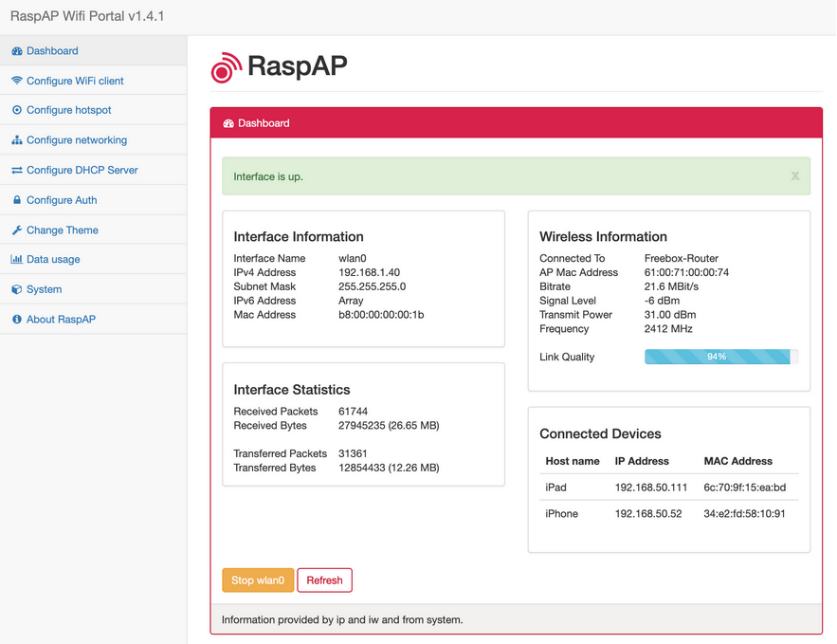


Figure  : DHCP configuration phone RaspAP

Figure  : Dashboard PC RaspAP

# Gestion des utilisateurs

## La base de données

MySQL est un système de gestion de base de données relationnelle.  Il est distribué sous une double licence [GPL](https://fr.wikipedia.org/wiki/Licence_publique_g%C3%A9n%C3%A9rale_GNU) et [propriétaire](https://fr.wikipedia.org/wiki/Logiciel_propri%C3%A9taire). Il fait partie des logiciels de gestion de [base de données](https://fr.wikipedia.org/wiki/Base_de_donn%C3%A9es) les plus utilisés au monde, autant par le grand public (applications web principalement) que par des professionnels.

## Docker et MySQL

Il existe des images MySQL Docker, mais elles ne sont pas compatibles avec Raspberry Pi car Raspbian utilise l’architecture arm32v7/ armel. Il existe également des images faites par la communauté mais n’étant pas officiel nous avons décidé de créer notre propre image MySQL.  
Nous sommes parti sur la base d’une image Debian Stretch. Dans un premier temps nous allons mettre à jour le Debian.

**RUN apt-get -y update && apt-get -y upgrade**

### Dossier avec les scripts SQL

Nous allons créer un dossier dans l’image qui contiendra les scripts SQL permettant d’initialiser les tables.

RUN mkdir /docker\_entrypoint-initdb.d/

Nous les utiliserons par la suite pour créer les tables dans la base de données à chaque création d’un container. Il faut également copier trouver les fichiers à transférer dans l’image.

COPY ./sql-scripts /docker\_entrypoint-initdb.d/ /docker\_entrypoint-initdb.d/

Un utilisateur est caractérisé par :

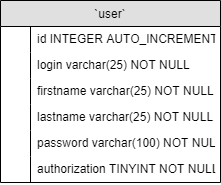
* Un id permettant d’identifier de manière unique l’utilisateur
* Un login permettant l’identification de l’utilisateur
* Son prénom
* Son nom
* Son mot de passe
* Ses droits

Figure  : Table 'user'

### Installation et configuration MySQL

Il faut ensuite installer mysql-server dans le container.

**RUN apt-get install -y mysql-server**

Nous allons maintenant créer un script de configuration qui sera lancé à la création du container. Nous allons commencer par définir le mot de passe de l’utilisateur root qui sera entré à la création du container.

mysql -u root --password=password -e "SET PASSWORD FOR 'root'@'localhost' = PASSWORD('"$MYSQL\_ROOT\_PASSWORD"');"

Nous devons ensuite créer notre base de données et la base de données dédiée aux tests unitaires.

**mysql -u root --password="$MYSQL\_ROOT\_PASSWORD" -e "CREATE DATABASE pi\_home;"**

**mysql -u root --password="$MYSQL\_ROOT\_PASSWORD" -e "CREATE DATABASE pi\_home\_test;"**

Souvenons-nous que nous mettons les scripts de création de tables dans le dossier « /docker\_entrypoint-initdb.d ». Afin d’exécuter ces scripts, nous allons récupérer les fichiers contenus dans le dossier et les exécuter une à une.

**for f in /docker\_entrypoint-initdb.d/\*; do**

**mysql -u root --password="$MYSQL\_ROOT\_PASSWORD" < $f**

**done**

Nous devons ensuite permettre la connexion en réseau local sur la base de données. Il faut changer la valeur de bind-address dans /etc/mysql/my.cnf par bind-address=0.0.0.0.

**sed -e "s/^bind-address\(.\*\)=.\*/bind-address=0.0.0.0/" -i /etc/mysql/my.cnf**

Enfin, il faut indiquer à l’image qu’il doit exécuter le script de configuration. Le script doit donc être copier, rendu exécutable puis exécuter.

**COPY ./docker-entrypoint.sh /usr/bin/**

**RUN chmod +x /usr/bin/docker-entrypoint.sh**

**RUN ./usr/bin/docker-entrypoint.sh**

### Construction de l’image et lancer le container comme démon.

Grâce à la commande CMD nous pouvons indiquer à l’image qu’il doit lancer le démon mysqld

**CMD ["mysqld"]**Le dockerfile étant terminé nous pouvons construire l’image grâce à la commande docker build et le paramètre -t qui permet de donner un nom à l’image. Le paramètre. (Point) permet d’indiquer que le dockerfile se trouve dans le dossier courant.

**docker build -t "pi\_home\_mysql" .**

Nous pouvons enfin lancer le container avec la commande docker run et en indiquant certain nombre de paramètres :

* --name : permet d’indiquer le nom du container.
* --net host : permet la communication en réseau local.
* -e : permet de donner des variables d’environnement. Dans notre cas nous donnons le mot de passe de root.
* -d : permet de lancer le container en démon.
* Le dernier paramètre doit être le nom de l’image permettant la création du container.

## Utiliser la base de données en python

Nous allons créer le fichier UserImpl qui contiendra toutes les fonctions permettant de manipuler la table ‘user’. UserImpl utilise la librairie pymysql pour manipuler la base de données. La dernière version stable peut être installer avec **pip**.

**pip install PyMySQL**

Nous pouvons nous connecter à la base de données en utilisant la fonction **connect** de la librairie. Nous devons entrer en paramètres :

* L’IP de la machine qui héberge la base de données
* L’utilisateur
* Le mot de passe
* Le nom de la base de données

**db = pymysql.connect(self.ip,self.user,self.password,self.database)**

Après la connexion, il faut récupérer un « cursor » qui va permettre d’exécuter les requêtes sql.

**cursor = db.cursor()**

La requête pourra alors être exécuter avec la fonction **execute**.

**cursor.execute(sql)**

Chaque manipulation de la connexion doit être un bloc **try** afin de pouvoir fermer la connexion quoiqu’il arrive dans un bloc **finally**.

**db = pymysql.connect(self.ip,self.user,self.password,self.database)**  
**try:**

**cursor = db.cursor()  
 cursor.execute(sql)  
finally:**

**db.close()**

Lorsque nous utilisons une requête **SELECT**, il est possible de récupérer toute les lignes avec la fonction **fetchall**. Nous obtenons alors un tableau contenant tous les utilisateurs. Chaque cellule du tableau est également un tableau représentant les colonnes dans la base de données. Il faut donc parcourir chaque ligne et récupérer les données pour créer un utilisateur.

**for row in results:**

**id = row[0]**

**login = row[1]**

**firstname = row[2]**

**lastname = row[3]**

**password = row[4]**

**authorization = row[5]**

**user = User(id, login, firstname, lastname, password, authorization)**

Toutes les requêtes qui modifient la base de données doivent être suivit de la fonction **commit** qui permet de valider les modifications sur la base de données. Si cette fonction n’est pas appelée, aucune modification ne sera prise en compte.

**db.commit()**

|  |  |  |
| --- | --- | --- |
| Fonction | Rôle | Requête |
| login | Permet de récupérer un utilisateur en fonction du login et du mot de passe | "SELECT \* FROM user where login = '"+ login + "' AND password = sha2('" + password + "salt',512)" |
| insert | Permet d’ajouter un utilisateur à la base de données | INSERT INTO user (login, firstname, lastname, password, authorization) VALUES (“login”,”firstaname”,”lastname”,”sha2(“passwordsalt',512),authorization) |
| select | Permet de récupérer un utilisateur en fonction de son id | SELECT \* FROM user WHERE id = id” |
| selectAll | Permet de récupérer tous les utilisateurs | SELECT \* FROM user |
| getUserByLogin | Permet de récupérer un utilisateur en fonction du login | SELECT \* FROM user where login = '" + login + "' |

|  |  |  |
| --- | --- | --- |
| updateUser | Permet de mettre à jour un utilisateur en fonction de son id | UPDATE user SET login ='" + user.getLogin() + "'" \  + " , firstname ='" + user.getFirstname() + "'" \  + " , lastname = '" + user.getLastname() + "'" \  + " , password = '" + user.getPassword() + "'" \  + " , authorization = " + str(user.getAuthorization()) \  + " WHERE id = " + str(id) |
| delete | Permet de supprimer un utilisateur en fonction de son id | sql = "DELETE FROM " + table + " WHERE id = " + str(id) |

## Flask

Flask est un microframework permettant de créer une api web en python. Le « micro » dans le microframework signifie que Flask vise à garder le noyau simple mais extensible. Flask ne prendra pas beaucoup de décisions pour nous, telles que la base de données à utiliser. Tout dépend de nous, de sorte que Flask puisse répondre à tous vos besoins et à tous vos besoins.

Flask peut être installer avec pip.

**pip install Flask**

### Création de l’API

La configuration minimale d’une application Flask est :

**from flask import Flask**

**app = Flask(\_\_name\_\_)**

Tout d’abord la classe Flaks est importé. Ensuite, nous créons une instance de cette classe.  
Nous devons également définir la fonction main lorsque le script est exécuté. Nous allons utiliser la fonction run de notre instance avec la valeur 0.0.0.0 pour l’argument host avec de pouvoir communiquer avec l’API sur le réseau local et la valeur 5001 pour le port.

**if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(host='0.0.0.0', port=5001)**

### 

### Flask-RESTful

**Flask-RESTful** est une extension pour Flask qui prend en charge la création rapide d’API REST. Nous allons utiliser afin de mieux structurer notre dossier. **Flask-RESTful.** Il peut être installer avec pip.

**pip install flask-restful**

### Création de la ressource utilisateur

Les principaux composants fournis par Flask-RESTful sont des ressources. Les ressources sont construites au-dessus des [vues connectables Flask](http://flask.pocoo.org/docs/views/) , nous permettant d'accéder facilement à plusieurs méthodes HTTP simplement en définissant des méthodes sur notre ressource.  
Nous allons créer la classe **UserRessource** qui hérite de **Ressource**. Cette première ressource sera une ressource CRUD. On aura alors les fonctions :

* get : permettant de récupérer un utilisateur si un id est défini ou tous les utilisateurs si aucun id n’est transmit
* post : permettant d’ajouter un utilisateur
* put : permettant de modifier un utilisateur
* delete : permettant de supprimer un utilisateur

Avant de programmer correctement toutes les fonctionnalités, nous allons d’abord comprendre comment ça marche. Nous allons commencer par créer la méthode get qui retourne seulement la chaine de caractère « hello world ».

**def get(self):  
 return "hello world"**

Il faut maintenant assigner une url à notre ressource grâce à la fonction **add\_ressource** de l’objet API. Nous indiquerons en paramètres la ressource et le chemin.

**api.add\_resource(UserResource, "/api/user")**

On peut alors tester d’accéder à notre fonction grâce à postman sur l’url **http://ip:5001/api/user. Postman**qui permet **d'envoyer toutes sortes de requêtes et les personnaliser très finement**. Il permet également de gérer l'authentification, les scripts de tests, etc. Postman sera utile par la suite pour vérifier les méthode POST, PUT et DELETE qui ne sont pas possible tester simplement via le navigateur comme pour une simple requête GET.Une fois la méthode GET sélectionner, l’url indiquer et l’appuis sur le bouton « Send » dans postman, on obtient alors notre chaine de caractères.

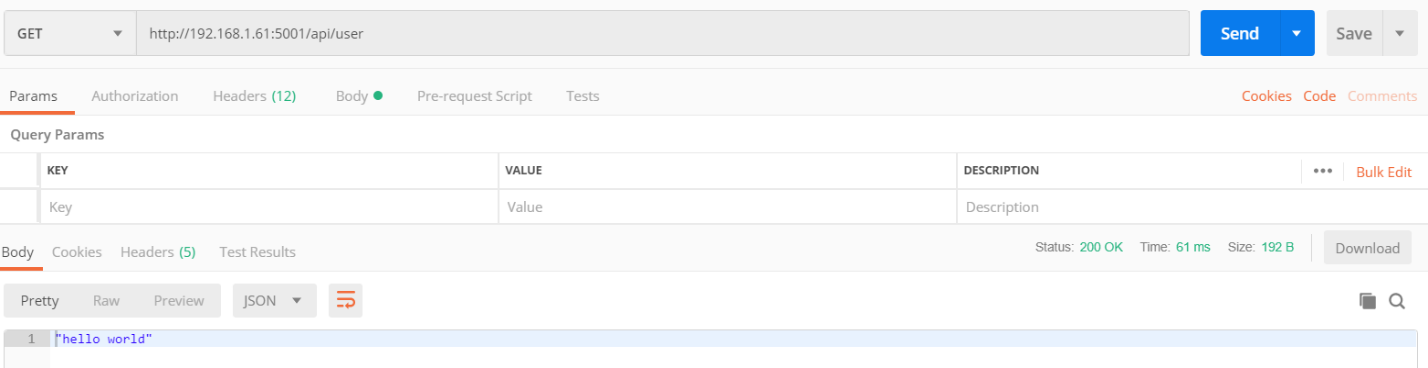


Figure : Requête Get retournant "Hello World"

### Login

Le problème de notre fonction get est qu’elle accessible par n’importe qui. Avant de programmer correctement nos fonctions permettant de gérer les utilisateurs, nous allons mettre en place un système d’authentification avec un jeton d’authentification. Un jeton d’authentification qui représente un utilisateur durant un laps de temps déterminé. Plutôt que de faire transiter le nom d'utilisateur ainsi que le mot de passe, c'est une chaîne de caractères délivrée par le serveur en charge de la gestion des utilisateurs que nous allons envoyer. Nous devons importer **flask\_jwt\_extended** dans **run.py.** Ensuite, il faut configurer la clé secrète JWT et initialisez JWT en ajoutant notre instance **app** à **JWTManager**.

**from flask\_jwt\_extended import JWTManager**  
**app.config ['JWT\_SECRET\_KEY'] = 'jwt-secret-string'**  
**jwt = JWTManager (app)**

Nous allons commencer par créer la ressource **LoginRessource** qui contiendra une fonction post qui va permettre de s’authentifier. Pour se connecter il faudra envoyer un login et mot de passe afin de faire une recherche dans la base de données. Nous enverrons ces données en format JSON. Par exemple pour s’authentifier avec le login Maxime et le mot de passe Test123\* nous devrons envoyer le JSON suivant :

**{  
 "login": "Maxime"  
 "password": "Test123\*"  
}**

Le JSON peut être récupéré par la fonction grâce à **request.get\_json().** Nous obtenons alors une sorte de dictionnaire et nous pourrons récupérer les valeurs en utilisant les clés « login » et « password ». On peut donc utiliser la fonction login de UserImpl afin de vérifier que nous obtenons un résultat dans la base de données

**content = request.get\_json()  
userImpl = UserImpl()  
user = userImpl.login(content["login"],content["password"])**

Si aucun résultat n’est trouvé, nous allons retourner un message d’erreur avec le code http 400.

**if(user == None):  
 response = jsonify('Login or password incorrect')  
 response.status\_code = 400  
 return response**

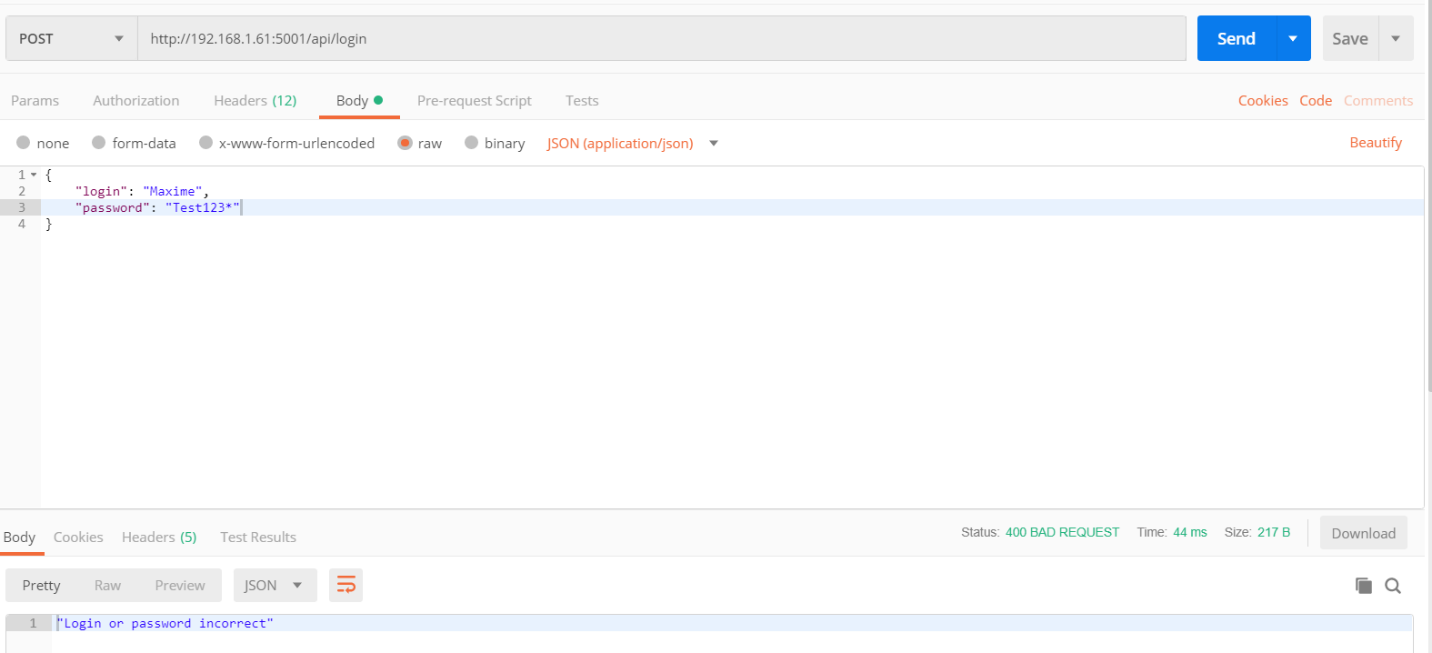


Figure 6 : Exemple d'un échec d'authentification

En cas de connexion réussie, nous retournerons deux jetons :

* Le jeton d’accès
* Le jeton d’actualisation

Pour la génération de jetons, nous utilisons deux fonction : **create\_access\_token** et **create\_refresh\_token**. Chaque fonction accepte au moins l’argument identity dans lequel nous allons passer l’id de l’utilisateur.

**access\_token = create\_access\_token(identity = str(user[0].getId()))  
 refresh\_token = create\_refresh\_token(identity = str(user[0].getId()))  
 return{  
 'message': 'Logged in as ' + str(user[0].getLogin()),  
 'access\_token':access\_token,  
 'refresh\_token':refresh\_token  
 }**

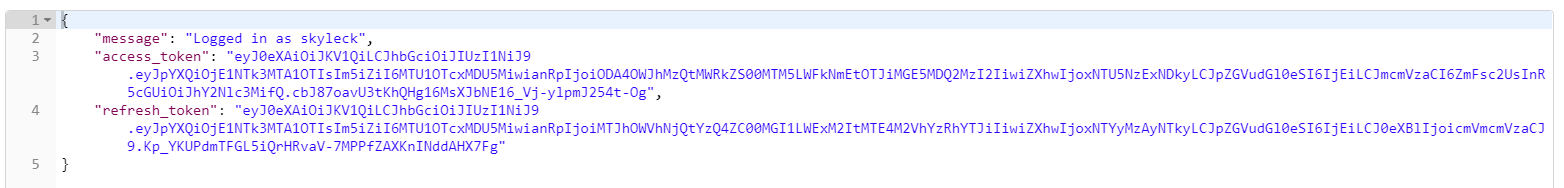


Figure 7 : Exemple d'authentification réussie

Nous pouvons maintenant protéger notre fonction get en ajoutant **@jwt\_required**.

**@jwt\_required  
def get(self,id=None):  
 return "hello world"**

Si nous essayons maintenant de contacter la méthode get, nous obtenons le message d’erreur **Missing Authorization Header**.

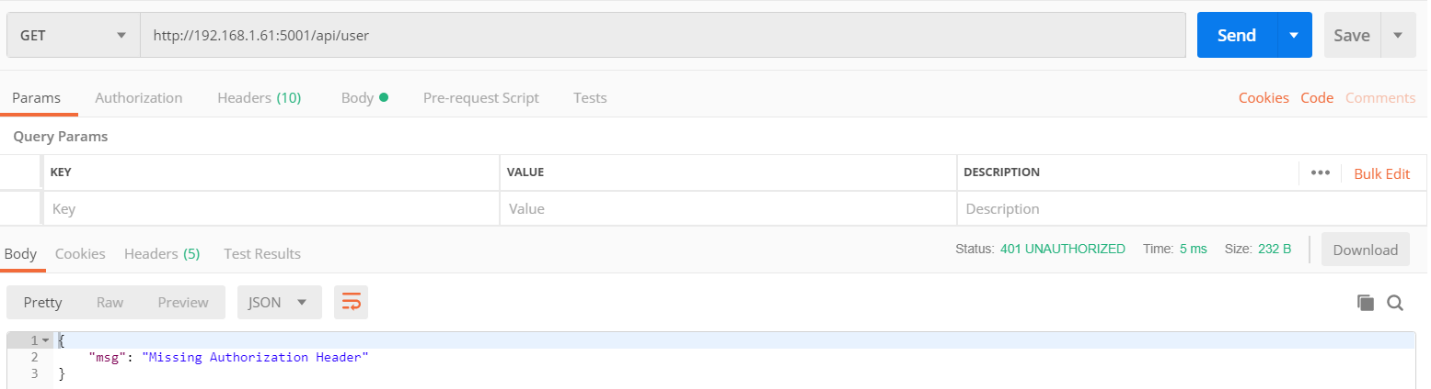


Figure 8 : Exemple d'erreur d'absence de jeton

Pour pouvoir utiliser la méthode, il faut ajouter la clef Authorizationavec le jeton comme valeur sous la forme **Bearer <jwt>**.

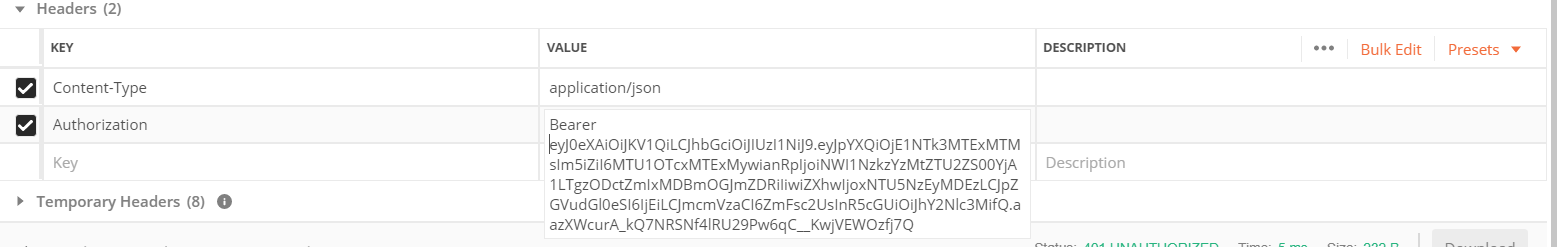


Figure 9 : Exemple d'header avec un jeton d'authentification

### CRUD

#### GET

Maintenant que nous pouvons nous connecter, nous allons programmer nos fonctions permettant de gérer les utilisateurs. Nous allons commencer par modifier notre méthode get pour qu’elle retourne un utilisateur ou tous les utilisateurs. Il faut récupérer l’utilisateur courant afin de pouvoir récupérer ses autorisations par la suite.

**curent\_user = self.userImpl.select(get\_jwt\_identity())**

Seulement un utilisateur autoriser pourra récupérer un ou les utilisateurs, un message doit être retourner si un utilisateur n’est pas autoriser.

**response = jsonify({'msg': 'Not authorize'})  
response.status\_code = 400  
return response**

Seulement un administrateur pourra récupérer tous les utilisateurs.  
  
**if(curent\_user[0].getAuthorization() == 2):  
 users = self.userImpl.selectAll()  
 return jsonify([e.jsonFormat() for e in users])**

Dans le temps de la récupération d’un utilisateur grâce à l’id, un utilisateur pourra aussi récupérer ses informations malgré qu’il ne soit pas administrateur.  
  
**if curent\_user[0].getAuthorization() == 2 or curent\_user[0].getId() == int(id):  
 users = self.userImpl.select(id)  
 return jsonify([e.jsonFormat() for e in users])**

#### POST

Cette fonction va permettre d’ajouter un utilisateur. Elle va dans un premier temps vérifier que les valeurs login,firstname,lastname ,password et confirmPassword ne sont pas vide. Si une valeur est vide un message d’erreur et la liste des champs vide sera retourner.

**content = request.get\_json()  
 inputVoid = []  
 if content["login"] == "":  
 inputVoid.append("login")  
 if content["firstname"] == "":  
 inputVoid.append("firstname")  
 if content["lastname"] == "":  
 inputVoid.append("lastname")  
 if content["password"] == "":   
 inputVoid.append("password")  
 if content["confirmPassword"] == "":  
 inputVoid.append("confirmPassword")  
 if len(inputVoid) != 0:  
 response = jsonify({'msg': 'One or more fields are not completed', 'column':  
 inputVoid})  
 response.status\_code = 400  
 return response**

Elle va ensuite que les valeurs password et confirmPassword sont égaux.

**if content["password"] != content["confirmPassword"]:  
 inputVoid.append("password")  
 inputVoid.append("confirmPassword")  
 response = jsonify({'msg': 'Passwords not equals', 'column': inputVoid})  
 response.status\_code = 400  
 return response**

Enfin, elle va vérifier que login n’existe pas.

**checkLogin = self.userImpl.getUserByLogin(content["login"])  
 if checkLogin is not None:  
 inputVoid.append("login")  
 response = jsonify({'msg': 'Login already exist !', 'column': inputVoid})  
 response.status\_code = 400  
 return response**

Après ces vérifications, l’utilisateur pourra être ajouter.  
  
**user=User(0,content["login"],content["firstname"],content["lastname"],content["password"],content["authorization"])  
self.userImpl.insert(user)  
return "User added"**

Attention, cette méthode ayant pour but de créer un utilisateur, elle doit être accessible par un utilisateur non authentifier afin de pouvoir de créer un compte.

#### PUT

La fonction put va servir à mettre à jour un utilisateur. Elle effectuera les mêmes vérifications de champs vide et de mot de passe que la fonction POST. Après ces vérifications, l’utilisateur peut être modifier.

**user=User(content["id"],content["login"],content["firstname"],content["lastname"],content["password"],content["authorization"])  
 self.userImpl.updateUser(content["id"],user)  
 return "User updated"**

Seulement un administrateur ou l’utilisateur concerner peut modifier ses informations.

#### DELETE

La fonction delete va servir à supprimer un utilisateur. Seulement un administrateur ou l’utilisateur concerner peut supprimer un compte.

**@jwt\_required  
 def delete(self):  
 curent\_user = self.userImpl.select(get\_jwt\_identity())  
 content = request.get\_json()  
 if curent\_user[0].getAuthorization() == 2 or curent\_user[0].getId() == int(content["id"]):  
 content = request.get\_json()  
 self.userImpl.delete(content["id"])  
 return "User deleted"  
 else:  
 response = jsonify({'msg': 'Not authorize'})  
 response.status\_code = 400  
 return response**

Conclusion

Bibliographie

<https://www.ass-security.fr/blog/protocole-mqtt-iot-objets-connectes-cyberattaque/>

<https://openest.io/2019/02/06/chiffrement-communication-mqtt-tls-ssl-mosquitto-et-paho/>

https://github.com/mqtt/mqtt.github.io/wiki

https://www.lemagit.fr/conseil/Internet-des-Objets-bien-comprendre-MQTT

https://www.ibm.com/support/knowledgecenter/fr/SSFKSJ\_7.5.0/com.ibm.mm.tc.doc/tc00000\_.htm

https://www.lebigdata.fr/mqtt-tout-savoir

https://openest.io/2018/05/22/mqtt-un-protocole-de-communication-pour-vos-objets-connectes/

<https://www.freecodecamp.org/news/docker-entrypoint-cmd-dockerfile-best-practices-abc591c30e21/>

<https://tecadmin.net/install-mysql-server-on-debian9-stretch/>

<https://wiki.debian-fr.xyz/Changer_le_mot_de_passe_mysql>

<https://pymysql.readthedocs.io/en/latest/user/index.html>

<https://www.w3schools.com/python/python_try_except.asp>

<https://codeburst.io/jwt-authorization-in-flask-c63c1acf4eeb>

<https://flask-restful.readthedocs.io/en/0.3.5/index.html>

<https://openclassrooms.com/fr/courses/4668056-construisez-des-microservices/5123020-testez-votre-api-grace-a-postman>

<https://www.getpostman.com/>

Lexique

Publisher :

Broker :

Subscriber :