

IoT application με χρήση Raspberry pi και integration me discord bot.

Ορέστης Νικόλας, 228438, 6^ο έτος

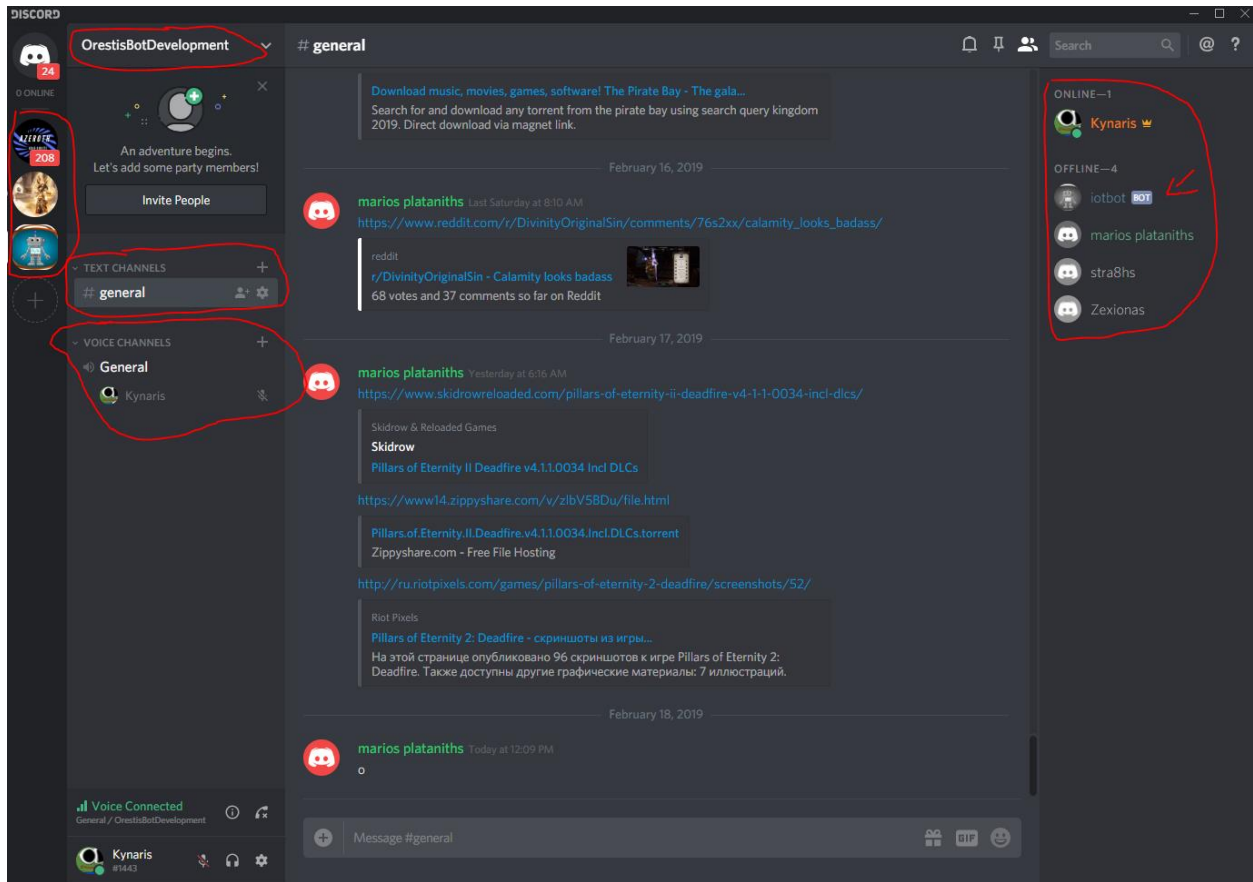
Εισαγωγή & Στόχοι της εργασίας:

Ξεκίνησα αυτή την εργασία προσπαθώντας να βρω μία λύση στο εξής πρόβλημα: Βρίσκομαι εκτός σπιτιού και θέλω με κάποιο τρόπο να:

- Έχω πρόσβαση σε διάφορες τιμές Sensors(Θερμοκρασίας, Πίεσης, Υγρασίας κτλ.) που βρίσκονται στο σπίτι μου
- Να μπορώ να ανοίγω/κλείνω τα φώτα του σπιτιού μου
- Να έχω έναν αυτοματισμό(σε μορφή actuation*) όπου σε περίπτωση λήψης μεγάλης θερμοκρασίας από τους σένσορες όπου υπάρχει καθαρή ένδειξη πυρκαγιάς να μου έρχεται ειδοποίηση άμεσα στο κινητό

Όλα τα παραπάνω θα γίνονται μέσω της αλληλεπίδρασης μου με ένα Bot μέσω της πλατφόρμας του Discord. Παρακάτω θα εξηγήσω ένα ένα ότι χρειάζεται για τη πλήρη κατανόηση της εργασίας.

To Discord App:



Όπως φαίνεται στη παραπάνω εικόνα, το Discord είναι ένα application πολύ παρόμοιο με το Skype. Το App αυτό πέρα από Windows είναι διαθέσιμο σε όλα τα mainstream λειτουργικά όπως Android, IOS, MAC, Linux κτλ. Το use case της συγκεκριμένης εργασίας είναι ότι θα χρησιμοποιώ το android discord application στο κινητό μου.

Ιστορικά, το κίνητρο για τη δημιουργία του Discord ήταν η ανάγκη της αγοράς για ένα αξιόπιστο App το οποίο θα είχε ως αρχικό κοινό το online gaming community με σκοπό την επίλυση του voice communication μεταξύ gamers.

Στα highlights της εικόνας διακρίνουμε τα εξής:

- Πάνω αριστερά βλέπουμε ότι βρισκόμαστε στον OrestisBotDevelopment server, τον οποίο έχω φτιάξει εγώ. Ο Server αυτός δεν τρέχει locally στον υπολογιστή μου απλά υπάρχει μια διαδικασία όπου έπρεπε να ακολουθήσω ώστε να δημιουργηθεί αυτός ο server και εγώ απλά είμαι ο admin και έχω full control και accessibility σε οτιδήποτε σχετίζεται με τον server αυτόν -από τα άτομα που μπορούν να συνδεθούν μέχρι πόσα κανάλια voice ή text θα έχει-
- Κάτω αριστερά από αυτό βλέπουμε μικρά εικονίδια τα οποία είναι όλοι οι Discord Server στους οποίους έχω συνδεθεί, στη συγκεκριμένη περίπτωση π.χ. βλέπουμε ότι

είμαι συνδεδεμένος σε 3 Server συνολικά. Για να συνδεθείς σε κάποιο server πρέπει να σου στείλει κάποιος ένα invite link αλλιώς δεν γίνεται να συνδεθείς.

- Μπορούμε επίσης να δούμε και τα text/voice channels τα οποία υπάρχουν στο συγκεκριμένο Server π.χ. Στον OrestisBotDevelopment βλέπουμε ότι έχω ένα general text channel και ένα general voice channel. Προφανώς ο κάθε Server μπορεί να έχει περισσότερα από ένα απλά για τους σκοπούς της εργασίας δε χρειάστηκε.
- Πάνω αριστερά βρίσκονται όλοι οι user που έχουν συνδεθεί στο Server(ανεξαρτήτως του αν είναι online ή όχι). Σημαντικό είναι να προσέξουμε ότι υπάρχει ένας user iotbot που έχει το tag [BOT] δίπλα του. Εγώ είμαι ο user Kynaris που έχει και τη πορτοκαλί κορόνα από δίπλα που σημαίνει ότι είμαι ο admin του server.

Τι είναι το Bot και πώς λειτουργεί:

Υπάρχει μια στάνταρ διαδικασία μέσω της οποίας δημιουργείς και κάνεις register ένα bot σε κάποιο server αλλά για να μη μπαίνω σε πολλές λεπτομέρειες θα αναφέρω μόνο ότι σχετίζεται με το σκοπό του project.

Το bot είναι ένας εικονικός χρήστης που έχει όλες τις ικανότητες ενός πραγματικού χρήστη(π.χ. έχει δικό του ID, tag, μπορεί να στέλνει μηνύματα κτλ.). Για τον έλεγχο του iotbot χρειάζεται internet connection, μια γλώσσα προγραμματισμού, το discord API που είναι free και το Bot token όπως φαίνεται παρακάτω.

```
class BotThread(threading.Thread):
    def __init__(self, threadID, name):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name

    def run(self):
        client.run("NDM5NDk1MjA1ODc2NTMxMjIw.DcUS7g.SlWfniUayYDXRkUgVqhPO5mlfW8")
```

Άρα προφανώς για να μπορεί να ελέγξει κάποιος το Bot μου χρειάζεται να γνωρίζει το παραπάνω hash. Παρόλα αυτά, υπάρχουν πολλά πρωτόκολλα security που χρησιμοποιεί το discord όπως το OAuth2 όπου εξασφαλίζουν αποκλειστικότητα στον έλεγχο του Bot ώστε ακόμα και να γνωρίζει κάποιος το Bot token να μη μπορεί να έχει έλεγχο.

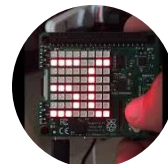
Με λίγα λόγια, το bot είναι ένας χρήστης όπου μπορείς μέσω του API να το προγραμματίσεις να κάνει ότι θες, από το να κάνει κάποιες απλές λειτουργίες όπως να απαντάει σε συγκεκριμένα μηνύματα με συγκεκριμένο τρόπο μέχρι τη δημιουργία ενός advanced AI με ικανότητες voice/speech recognition, image processing κτλ.

Γενικότερα τα τελευταία χρόνια υπάρχει μεγάλο ενδιαφέρον σε bot AI development. Ακόμα και η ίδια η εταιρία Discord κάνει ανά χρονικά διαστήματα διαγωνισμούς για καλά AI τα οποία π.χ. έχουν δυνατότητες όπως να σου στέλνουν σε PM(personal message) καθημερινά τις ειδήσεις αναλόγως των ιστοσελίδων που έχεις επισκεφθεί τελευταία κτλ.

Για τη συγκεκριμένη εργασία το μόνο που μας ενδιαφέρει είναι το **iotbot** να μπορεί να ανταποκρίνεται σε απλές εντολές μέσω PM γιατί πρακτικά η επικοινωνία(για τη γνώση της θερμοκρασίας, για να ανοίξουν τα φώτα κτλ.) θα είναι μέσω PM με ένα χρήστη(π.χ. εμένα) και το **iotbot**. Σημαντικό είναι να σημειώσουμε ότι ο οποιοσδήποτε έχει το tag του **iotbot**(**iotbot#0362**) μπορεί να αλληλεπιδράσει μαζί του ανεξαρτήτως του αν βρίσκονται στον ίδιο server ή όχι.

Τι είναι το Raspberry pi και ποια η χρησιμότητά του στο Project:

Το Pi είναι ένας credit-card-sized μικροϋπολογιστής με σχετικά περιορισμένες υπολογιστικές δυνατότητες. Το Pi 3 b+ που έχω διαθέτει 4 Core @ 1.4 GHz processor και 1 Gb RAM. Έχει το δικό του linuxoid λογισμικό που λέγεται **Rasbian** και μπορεί να πει κάποιος ότι είναι πολύ παρόμοιο με το **Arduino**. Έχει **WiFi**, θύρες για ethernet ,HDMI , 4x USB2, microSD κτλ. Ανεξαρτήτως των σχετικά περιορισμένων πόρων του, το **Rasbian** λειτουργικό το καθιστά ένα fully functional υπολογιστή όπου μπορώ να κάνω Python development.



Η κύρια χρησιμότητα του έγκειται στο **Sense Hat**, που είναι ένα add-on board για το Pi και διαθέτει τους σένσορες θερμοκρασίας, πίεσης και υγρασίας. Πέρα από αυτά, όπως φαίνεται στην παραπάνω εικόνα, διαθέτει και μια οθόνη 8x8 led pixel την οποία έχω χρησιμοποιήσει για να κάνω το actuation που θα ανάβουν τα φώτα στο σπίτι (δεν είναι τα κανονικά φώτα του σπιτιού αλλά με τα εργαλεία που είχα στη διάθεση μου ήταν η καλύτερη προσέγγιση που θα μπορούσα να κάνω). Το **sense hat** έχει και πολλά άλλα διάφορα features αλλά δε μας ενδιαφέρουν σε αυτή την εργασία.

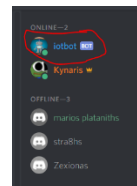
Χρησιμοποιώντας το Python module: **Sense_Hat** έχω πλήρη πρόσβαση στους σένσορες και στο led pixel screen

Πως συνδιάστηκαν όλα τα παραπάνω:

Εδώ θα αναφέρω σύντομα το τι γίνεται και αργότερα θα τα αναλύσω όλα σε βάθος. Αρχικά όπως προανέφερα, το **iotbot** χρειάζεται κάποιον κώδικα(ας τον ονομάσουμε **BotCode**) όπου θα ελέγχει τη συμπεριφορά του. Πέραν αυτού, ο **BotCode** θα πρέπει με κάποιο τρόπο να γνωρίζει τις τιμές των σενσόρων του Pi. Προφανώς θα χρειαστεί ένας 2^{ος} κώδικας ο οποίος θα τρέχει στο Pi(ας τον ονομάσουμε **PiCode**). Η λειτουργία του **PiCode** θα είναι να έχει την πλήρη πρόσβασή στους σένσορες μέσω του **Sense Hat** και έλεγχο του 8x8 pixel led screen. Είναι προφανές ότι η αποστολή μηνυμάτων μεταξύ των 2 αυτών κωδίκων είναι απαραίτητη για τους στόχους της εργασίας.

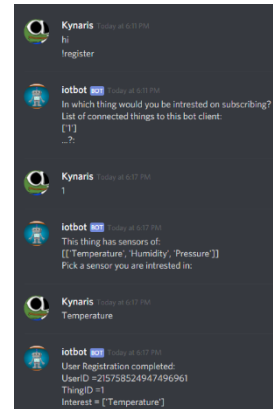
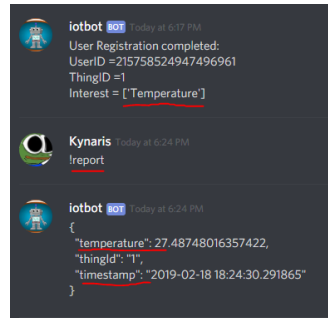
Συγκεκριμένα το **BotCode**:

- Τρέχει στο desktop μου ή στο laptop μου. Με το που ξεκινήσει βλέπουμε το **iotbot** online
- Μπορεί και κάνει parse οτιδήποτε στέλνει οποιοσδήποτε discord user μέσω PM στο **iotbot**. Συγκεκριμένα, άμα στείλω κάποια εντολή την οποία δεν αναγνωρίζει τότε απλά αγνοεί οτιδήποτε περιέχει το PM. Οι μόνες

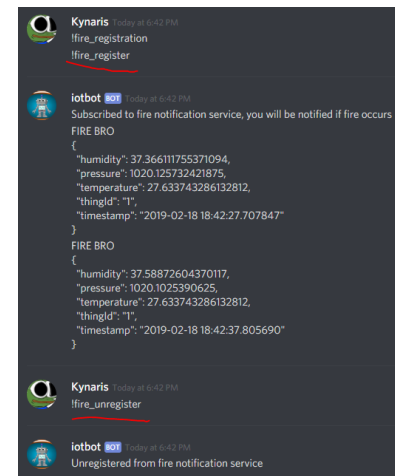
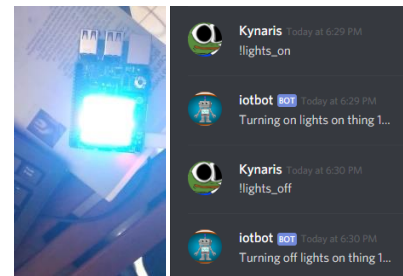


εντολές που αναγνωρίζει ο BotCode είναι !register, !report, !lights_on, !lights_off, !fire_register, !fire_unregister. Δεξιά βλέπουμε ότι στο hi δεν ανταποκρίνεται ενώ στο !register ανταποκρίνεται.

- Η εντολή !register χρησιμοποιείται σαν **εγγραφή ενός discord user πάνω στο bot**. Πρακτικά ο BotCode κρατάει αρχείο με διάφορες πληροφορίες όλων των εγγεγραμμένων χρηστών. Για να ανταποκριθεί σε οποιαδήποτε άλλη εντολή πρέπει πρώτα να έχει πραγματοποιηθεί το registration.
- Η εντολή !report δείχνει τις τελευταίες τιμές των σενσόρων αναλόγως το registration που έχει κάνει ο discord user. Π.χ. εδώ με ενδιέφερε μόνο η θερμοκρασία.



- Οι εντολές !lights_on και !lights_off κάνουν το BotCode να στείλει μήνυμα στο PiCode για να ανάψει το led screen.
- Τα !fire_register, !fire_unregister είναι σε περίπτωση όπου ο user θέλει να κάνει registration στο fire notification service. Άμα το bot λάβει κάποια εξωπραγματική τιμή θερμοκρασίας τότε στέλνει μήνυμα στους αντίστοιχους ενδιαφερόμενους ότι υπάρχει ένδειξη φωτιάς, όπως φαίνεται δεξιά. Προφανώς για να κάνω το demonstration εδώ άλλαξα τις τιμές που θερμοκρασίας που χρειάζεται να ελέγξει για να συμπεράνει φωτιά ώστε οι 27 βαθμοί του σπιτιού μου να αρκούν.



Συγκεκριμένα το PiCode:

- Κάνει register με το που ξεκινήσει το συγκεκριμένο “thing” στο bot. Όπως παρατηρήσαμε παραπάνω όταν ένας discord user θέλει να κάνει !register πάνω στο bot τότε τον ρωτάει ποιο ή ποια things τον ενδιαφέρουν. Η έννοια thing στο IoT είναι οποιοδήποτε σύστημα έχει πολύ μικρή υπολογιστική ικανότητα και διαθέτει κάποια sensors ή μπορεί να κάνει κάποια actuation. Προφανώς το Raspberry Pi δε μπορεί να θεωρηθεί thing αλλά για χάρη της εργασίας το θεωρώ. Το registration αυτό το έχω κάνει για λόγους scalability. Θεωρητικά το bot θα έχει πολλά things που θα είναι registered πάνω του και απλά οι discord users θα διαλέγουν ποια things τους ενδιαφέρουν και συγκεκριμένα ποιοι σένσορες και το bot κρατάει μια λίστα με όλες αυτές τις πληροφορίες και για τα discord user registration αλλά και για τα thing registration. Στο thing registration το κάθε thing πέρα από κάποιο unique ID που θα πρέπει να διαθέτει θα πρέπει να δίνει στο bot και πληροφορίες όπως το τί είδους σένσορες διαθέτει. Προφανώς εφόσον δεν είχα δεύτερο Raspberry Pi ούτε δεύτερο Sense Hat δεν μπορούσα να τεστάρω το σενάριο όπου θα είναι 2 things registered πάνω στο bot μου αλλά θεωρητικά ο κώδικας που έχω κάνει το χειρίζεται κανονικά. Η δεξιά εικόνα είναι το python console στο Raspberry αφού τρέξει το PiCode. Βλέπουμε ότι στέλνει το registration info στο BotCode.
- Το έχω προγραμματίσει ώστε να στέλνει τις τιμές των σενσόρων κάθε 10 sec. Άρα όταν ένας discord user στέλνει στο bot !report, το bot θα απαντήσει με τη πιο πρόσφατη τιμή που έλαβε από το συγκεκριμένο thing η οποία θα είναι το πολύ πριν 10 sec
- Τα LIGHTS ON BABY και lights off.. που βλέπουμε στην εικόνα είναι όταν έστειλα εγώ στο bot από τον discord user μου !lights_on, !lights_Off απλά πέρα από το ότι άναψε το led screen είναι μια ένδειξη ότι έλαβε το μήνυμα σωστά

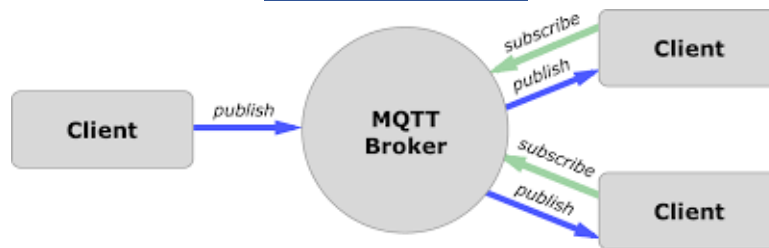
```
Python 3.5.3 (/usr/bin/python3)
>>> %Run piClient.py

connecting to broker... iot.eclipse.org
connected!
making registration to the bot client...
registration done!
connection established
reporting every 10 sec
LIGHTS ON BABY
lights off..
```

Άρα με λίγα λόγια το BotCode είναι υπεύθυνο για το πώς θα αλληλεπιδρά το iotbot με τους discord users και το PiCode για να στέλνει στο iotbot τις τιμές των σενσόρων και να ανάβει το led screen.

Παρακάτω θα αναλύσω το πώς επιτυγχάνεται η επικοινωνία μεταξύ των 2 αυτών κωδίκων, εφόσον ο ένας τρέχει στο laptop και ο άλλος τρέχει στο Pi.

MQTT protocol:



Το MQTT είναι ένα πρωτόκολλο επικοινωνίας που λειτουργεί πάνω από TCP/IP το οποίο χρησιμοποιείται κυρίως σε IoT εφαρμογές και τα κύρια χαρακτηριστικά του είναι μικρό code footprint και bandwidth usage. Το μοντέλο λειτουργίας είναι σχετικά απλό στην κατανόηση. Πρόκειται για publish/subscribe πρωτόκολλο και χρειάζεται κάποιον server(στο MQTT πρωτόκολλο οι Server ονομάζονται Brokers) για να επιτευχθεί η επικοινωνία μεταξύ 2 client.

Όσον αφορά το project αυτό, μπορούμε να φανταστούμε τους κώδικες BotCode και PiCode ότι πέρα από όλα αυτά που προανέφερα αποτελούν και MQTT clients και ο broker που χρησιμοποιούν για την ανταλλαγή μηνυμάτων μεταξύ τους είναι ο public broker: “iot.eclipse.org”. Δεξιά φαίνεται με το Python shell με το που τρέξει ο BotCode ότι συνδέεται σε αυτόν τον broker.

```
connecting to broker iot.eclipse.org
connection established
bot is ready!
new thing registration
{
  "sensors": [
    "Temperature",
    "Humidity",
    "Pressure"
  ],
  "thingId": "1"
}
```

```
class MqttThread(threading.Thread):
    def __init__(self, threadID, name):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name

    def run(self):
        broker_addr = "iot.eclipse.org"
        mqttclient = mqtt.Client("BotClient")
        mqttclient.on_connect = MqttOn_connect
        mqttclient.on_disconnect = MqttOn_disconnect
        mqttclient.on_message = MqttOn_message

        # connecting
        print("connecting to broker", broker_addr)
        mqttclient.connect(broker_addr)
        time.sleep(1)

        # BotClient listening reports & registration
        mqttclient.loop_start()
        mqttclient.subscribe("discord/!pm")
        mqttclient.subscribe("discord/thingRegistration")

        # BotClient publishing actuators routine
        listsize = len(actuation)
        while 1:
            #Polling to see if actuation list has changed and then publishing the most recent one
            if listsize != len(actuation):
                payload = pickle.dumps(actuation[-1])
                mqttclient.publish("discord/actuators", payload, 0)
                listsize = len(actuation)

            time.sleep(1)

        mqttclient.disconnect()
        mqttclient.loop_stop()

# Create new threads
thread1 = MqttThread(1, "Bot")
thread2 = MqttThread(2, "Mqtt")

# Start new Threads
thread1.start()
thread2.start()

while 1:
    MqttThread.run()
```

Αριστερά είναι το κομμάτι του κώδικα μέσα στο BotCode το οποίο είναι υπεύθυνο για το thread που διαχειρίζεται το MQTT client. Και οι 2 MQTT clients έχουν 2 συγκεκριμένες δουλειές που πρέπει να φέρουν εις πέρας.

1. Να κάνουν “subscribe” σε συγκεκριμένα και προκαθορισμένα topics τα οποία τους ενδιαφέρουν ώστε να λαμβάνουν μηνύματα που γίνονται “publish” σε αυτά.
2. Να κάνουν “publish” μηνύματα τα οποία πρέπει να στείλουν πάλι σε συγκεκριμένα και προκαθορισμένα topics.

Ένα παράδειγμα για το πως δουλεύει όλο αυτό είναι: Όταν ένας discord user στέλνει PM στο iotbot, !lights_on αυτό που γίνεται είναι ότι το BotCode αναγνωρίζει την εντολή και το MQTT

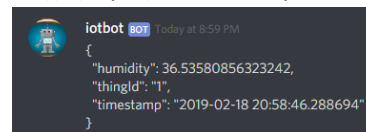
κομμάτι είναι υπεύθυνο να κάνει publish στο topic “discord/actuators” ένα συγκεκριμένο μήνυμα με συγκεκριμένη δομή το οποίο θα περιέχει όλη τη πληροφορία για το τι είδος actuation είναι, από ποιόν registered discord user έρχεται, σε ποιο thing αναφέρεται(target thingID) κτλ. Αντιστοίχως, το MQTT κομμάτι του PiCode είναι υπεύθυνο να κάνει “subscribe” στο ίδιο topic ώστε όταν σταλθεί το μήνυμα

να το κάνει parse, να αναγνωρίσει αν το actuation message αναφέρεται σε αυτό το thing και στην περίπτωση όπου πρέπει να υλοποιήσει το actuation να το φέρει εις πέρας και να ανάψει το led screen.

Ένα αντίστοιχο παράδειγμα με την αντίθετη ροή πληροφορίας είναι το thing registration όπου το PiCode πρέπει να κάνει register πάνω στο iotbot που προανέφερα. Στο συγκεκριμένο σενάριο ο PiCode με το που ξεκινήσει να τρέχει κάνει publish στο topic “discord/thingRegistration” τα registration info του και αντίστοιχα το BotCode επειδή είναι subscribed σε αυτό το topic θα λάβει το μήνυμα και θα εκπληρώσει το thing registration.

Ένα τρίτο παράδειγμα είναι στο !report. Πρακτικά εδώ το BotCode είναι ήδη subscribed στο “discord/inprt” topic και απλά κάθε 10 sec το PiCode κάνει publish σε αυτό το topic το report. Έτσι το BotClient λαμβάνει το report και το αποθηκεύει άρα για να απαντήσει στον discord user που του έστειλε !report το μόνο που έχει να κάνει είναι να στείλει τις τελευταίες τιμές που αποθήκευσε από αυτό το thing που είναι subscribed ο συγκεκριμένος discord user.

Είναι **σημαντικό** να τονίσουμε ότι όλα αυτά τα μηνύματα που στέλνονται μέσω MQTT έχουν συγκεκριμένη δομή. Για να το καταφέρω αυτό, έφτιαξα ένα **δικό μου API** (Application Interface) το οποίο γνωρίζει και το BotCode και το PiCode. Αυτό το API περιλαμβάνει τις κλάσεις Actuators, Report, ThingRegistration (Όπως φαίνεται και στους παραδοτέους κώδικες, και οι 2 φακελοι περιέχουν τα παραπάνω αρχεία). Όταν για παράδειγμα θέλει το PiCode να στείλει ένα report αυτό που κάνει είναι ότι φτιάχνει ένα object τύπου Report το οποίο έχει ως attributes τις αντίστοιχες τιμές των σενσόρων στα αντίστοιχα πεδία. Μετά αφού φτιάξει το object το κάνει serialize σε JSON μορφή όπως φαίνεται στην εικόνα δεξιά και **μετά** το κάνει publish στο “discord/inprt”. Το BotCode που προφανώς έχει κάνει ήδη subscribe σε αυτό το topic λαμβάνει το μήνυμα αλλά πρέπει πρώτα να το κάνει deserialize ώστε να καταφέρει να δημιουργήσει το ίδιο object και να το αποθηκεύσει. Το συγκεκριμένο API έπρεπε να το κάνω κυρίως για **2 λόγους**:

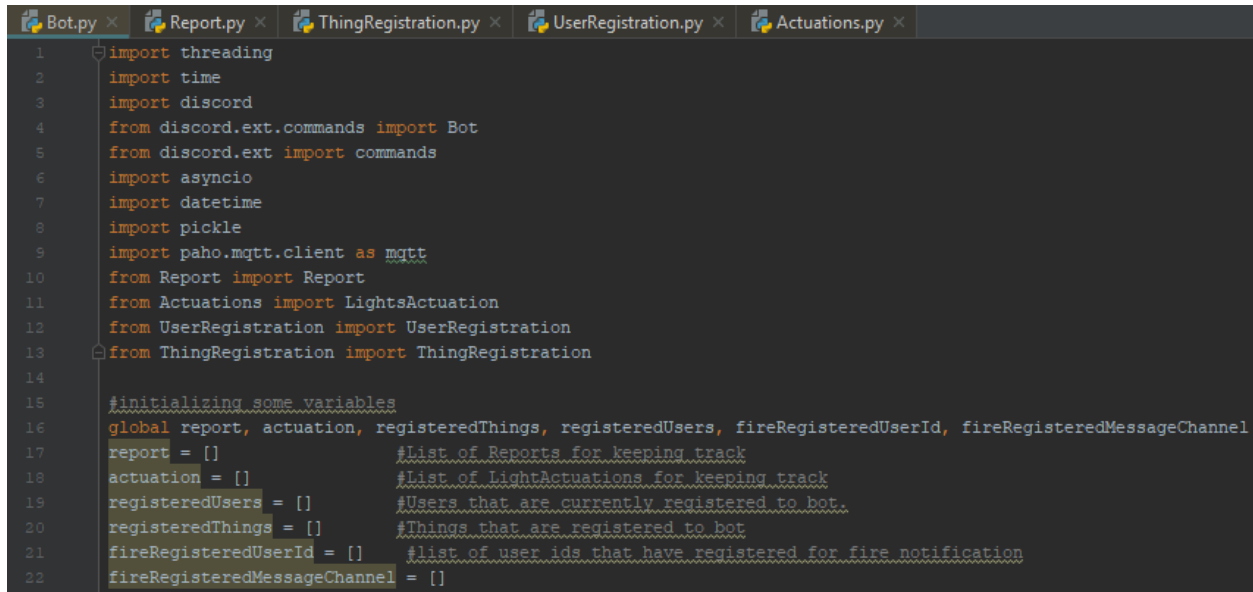


- **Security:** Επειδή ο MQTT broker είναι public αυτό σημαίνει ότι ο οποιοσδήποτε μπορεί να κάνει subscribe/publish στα topics που χρησιμοποιώ. Κάνοντας serialize τα δεδομένα μου ο επιτήδειος που τυχόν θα έχει κάνει subscribe και τα λαμβάνει δε θα μπορεί να καταλάβει τι είναι και θα βλέπει ότι λαμβάνει σκουπίδια. Άρα παρόλο που το ίδιο topic μπορεί να χρησιμοποιείται τυχόν και από άλλους τα μηνύματα που στέλνω θα τα καταλαβαίνουν μόνο το BotCode και το PiCode.
- **Αποφυγή λήψης σκουπιδιών:** Για παρόμοιο με το παραπάνω σενάριο, κάποιος μπορεί να κάνει publish πάνω σε ένα από τα topics που χρησιμοποιώ για την επικοινωνία μεταξύ BotCode και PiCode. Μολις λάβουν οι MQTT clients μου κάποιο μήνυμα προσπαθούν απευθείας να το κάνουν deserialize και να το αποθηκεύσουν σαν object. Άμα το μήνυμα που θα λάβουν δεν είναι κάποιο serialized object από τις συγκεκριμένες -γνωστές- κλάσεις που έχω στο API τότε το θεωρούν σκουπίδι και δε το λαμβάνουν υπ'οψιν.

Οι κώδικες:

Εδώ θα αναλύσω όσο μπορώ τα πιο σημαντικά κομμάτια των κωδίκων και το πως ακριβώς έχω κάνει την υλοποίηση. Αρχικά να αναφέρω ότι το development έγινε σε Python 3.6 και τα IDE που χρησιμοποίησα ήταν PyCharm όσον αφορά το BotCode και σε Thommy Python editor στο PiCode.

BotCode:



```
1 import threading
2 import time
3 import discord
4 from discord.ext.commands import Bot
5 from discord.ext import commands
6 import asyncio
7 import datetime
8 import pickle
9 import paho.mqtt.client as mqtt
10 from Report import Report
11 from Actuations import LightsActuation
12 from UserRegistration import UserRegistration
13 from ThingRegistration import ThingRegistration
14
15 #initializing some variables
16 global report, actuation, registeredThings, registeredUsers, fireRegisteredUserId, fireRegisteredMessageChannel
17 report = [] #List of Reports for keeping track
18 actuation = [] #List of LightActuations for keeping track
19 registeredUsers = [] #Users that are currently registered to bot.
20 registeredThings = [] #Things that are registered to bot
21 fireRegisteredUserId = [] #list of user ids that have registered for fire notification
22 fireRegisteredMessageChannel = []
```

Εδώ αρχικά κάνω import όλα τα απαιτούμενα modules. Βλέπουμε ότι κάνω import threading γιατί θέλω να τρέχει 2 παράλληλες διεργασίες το Bot μου, μία για να χειρίζεται την αλληλεπίδραση με τους discord users και μια για το κομμάτι επικοινωνίας MQTT. Το discord είναι για να χειρίζομαι το discord κομμάτι ενώ το paho.mqtt.client χρησιμοποιείται για το MQTT κομμάτι. Τα Report/Actuations/UserRegistration/ThingRegistration είναι στο API μου, το pickle είναι για να χειρίζομαι JSON δεδομένα(serialization/deserialization) και το asyncio είναι για ασύγχρονο αντικειμενοστραφή προγραμματισμό. Πέρα από αυτά βλέπουμε και διάφορες λίστες και μεταβλητές που χρειάζεται να αποθηκεύει το BotCode και τις αρχικοποιώ.

```

24
25 #MQTT functions
26 def MQTTon_connect(client, userdata, flags, rc):
27     if rc==0:
28         print("connection established")
29     else:
30         print("could not establish connection:", rc)
31
32 def MQTTon_disconnect(client, userdata, flags, rc=0):
33     print("Disconnected with result code: " + str(rc))
34
35 def MQTTon_message(client, userdata, msg):
36     temp = pickle.loads(msg.payload)
37     if type(temp) == Report:
38         #Checking if the id of the thing that reported is known(has registered)
39         if temp.thingId in [things.thingId for things in registeredThings]:
40             report.append(temp)
41             #print("report received:\n", report[-1])
42         else:
43             print("received report from unknown source")
44
45     if type(temp) == ThingRegistration:
46         registeredThings.append(temp)
47         print("new thing registration\n", registeredThings[-1])

```

Εδώ ορίζω τις συναρτήσεις που χρειάζομαι για το MQTT κομμάτι. Για παράδειγμα το MQTTon_message ορίζει το τι θα κάνει το BotCode όταν λάβει κάποιο μήνυμα σε κάποιο από τα topics που έχει κάνει subscribe.

```

49 #----- DISCORD PART -----
50 Client = discord.Client()
51 global client
52 client = commands.Bot(command_prefix="!")
53
54 async def fire_notification():...
55
56
57 @client.event
58 async def on_ready():...
59
60
61 @client.event
62 async def on_message(message):...
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216

```

Εδώ είναι που ξεκινάω τον κώδικα για το discord κομμάτι του bot. Σημαντικό να αναφέρω ότι πρόκειται αποκλειστικά για asynchronous event-driven προγραμματισμό γι' αυτό ορίζω events π.χ κάθε φορά που λαμβάνει κάποιο PM το iotbot στο discord τότε γίνεται trigger το on_message. Μέσα σε αυτές τις γραμμές(88-216) είναι που ορίζω πως ανταποκρίνεται το bot αναλόγως του τί του στάλθηκε σε PM και από ποιόν.

Βλέπουμε π.χ. στην παρακάτω εικόνα στις γραμμές 89-116 τί κάνει το iotbot σε περίπτωση που λάβει μήνυμα !register.

```

87 @client.event
88 async def on_message(message):
89     registeredUserIds = [x.userid for x in registeredUsers]
90
91     if message.content == "!register":
92         #Checking if user is new
93         if message.author.id not in registeredUserIds:
94
95             #Checking what things are connected to this bot client
96             thingIds = [tid.thingId for tid in registeredThings]
97             await client.send_message(message.channel, "In which things would you be interested on subscribing? List of connected things to this bot client:\n" + str(thingIds) + "\n...?")
98             reply = await client.wait_for_message(author=message.author, channel=message.channel)
99             targetThingId = reply.content
100
101             #Checking what sensors the selected thing has
102             ansrs = [x.sensors for x in registeredThings if x.thingId == targetThingId]
103             await client.send_message(message.channel, "This thing has sensors of:\n" + str(ansrs) + "\nPick a sensor you are interested in!")
104             reply = await client.wait_for_message(author=message.author, channel=message.channel)
105             targetSensor = reply.content
106             targetSensor = targetSensor.split()
107
108             #Checking if user input is valid
109             if targetThingId in [x.thingId for x in registeredThings] and set(targetSensor).issubset(["Temperature", "Humidity", "Pressure"]):
110                 #Finishing the registration
111                 newRegistration = UserRegistration(message.author.id, targetThingId, targetSensor)
112                 registeredUsers.append(newRegistration)
113                 await client.send_message(message.channel, "User Registration completed: \nUserID = " + str(message.author.id) + "\nThingID = " + str(targetThingId) + "\nInterest = " + str(targetSensor))
114             else:
115                 await client.send_message(message.channel, "Invalid registration input... aborting registration...")
116
117 #The user already has a registration and wants to make a new one
118 else:

```

Μετά από αυτά εφόσον έχω ορίσει τη συμπεριφορά του Discord Part του iotbot δημιουργώ το thread.

```

207 class BotThread(threading.Thread):
208     def __init__(self, threadID, name):
209         threading.Thread.__init__(self)
210         self.threadID = threadID
211         self.name = name
212
213     def run(self):
214         client.run("WDM5NDk1MjA1ODc2NTM4dYlw.DcUS7g.SiWfojUayYDxRkUgVghPO5mlfW8")
215

```

Παρακάτω είναι το MQTT κομμάτι του BotCode. Φτιάχνει το thread και ορίζει όλη τη ρουτίνα που αφορά τον MQTT client. Βλέπουμε ότι κάνει connect στον broker, κάνει τα MQTT callback function bindings με αυτές τις συναρτήσεις που είχαμε ορίσει στην αρχή του κώδικα, κάνει subscribe στα topics που θέλει να λαμβάνει μηνύματα("discord/inpt", "discord/thingRegistration") και κάνει publish όταν χρειάζεται στο "discord/actuators" αν χρειαστεί

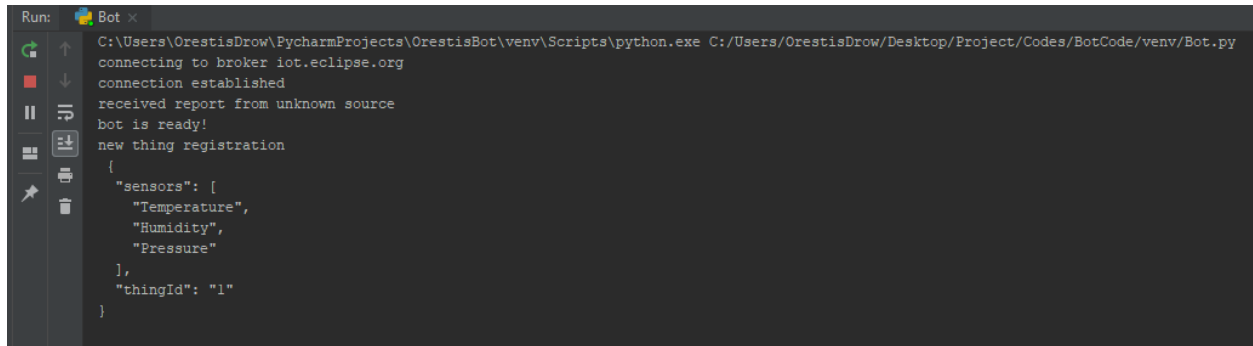
```

216 # ----- MQTT PART -----
217 class MqttThread(threading.Thread):
218     def __init__(self, threadID, name):
219         threading.Thread.__init__(self)
220         self.threadID = threadID
221         self.name = name
222
223     def run(self):
224         broker_addr = "mqtt://iotcloud.org"
225         MQTTclient = mqtt.Client("BotClient")
226         MQTTclient.on_connect = MQTTon_connect
227         MQTTclient.on_disconnect = MQTTon_disconnect
228         MQTTclient.on_message = MQTTon_message
229
230         # connecting
231         print("connecting to broker", broker_addr)
232         MQTTclient.connect(broker_addr)
233         time.sleep(1)
234
235         # BotClient listening reports & registrations
236         MQTTclient.loop_start()
237         MQTTclient.subscribe("discord/Inpt")
238         MQTTclient.subscribe("discord/thingRegistration")
239
240         # BotClient publishing actuators routine
241         listsize = len(actuation)
242         while 1:
243             #Polling to see if actuation list has changed and then publishing the most recent one
244             if listsize != len(actuation):
245                 payload = pickle.dumps(actuation[-1])
246                 MQTTclient.publish("discord/actuators", payload, 0)
247                 listsize = len(actuation)
248             time.sleep(1)
249
250         MQTTclient.disconnect()
251         MQTTclient.loop_stop()

```

```
266 # Create new thread
267 thread1 = BotThread(1, "Bot")
268 thread2 = MqttThread(2, "Mqtt")
269
270 # Start new Threads
271 thread1.start()
272 thread2.start()
273
274 while 1:
275     pass
```

Τέλος, τρέχει και τα 2 αυτά thread ταυτόχρονα μέχρι να γίνει kill το πρόγραμμα. Στο Python Console φαίνεται κάτι σαν και αυτό όταν τρέχει.



```
Run: Bot x
C:\Users\OrestisDrow\PycharmProjects\OrestisBot\venv\Scripts\python.exe C:/Users/OrestisDrow/Desktop/Project/Codes/BotCode/venv/Bot.py
connecting to broker iot.eclipse.org
connection established
received report from unknown source
bot is ready!
new thing registration
{
  "sensors": [
    "Temperature",
    "Humidity",
    "Pressure"
  ],
  "thingId": "1"
}
```

PiCode:

```
piClient.py
import time
import datetime
import paho.mqtt.client as mqtt
import random
from Report import Report
from sense_hat import SenseHat
import json
import pickle
from Actuators import LightsActuation
from ThingRegistration import ThingRegistration

global currState
currState = "0"
```

Στο PiCode τα imports είναι παρόμοια και για παρόμοιους λόγους. Η μεταβλητή currState είναι για να ξέρει αν η led οθόνη είναι on ή off. Κύρια διαφορά το module SenseHat όπου είναι για τον έλεγχο του Sense Hat. Η συνάρτηση GetSensorValues είναι καθαρά για να παίρνει ένα snapshot των τιμών στους σένσορες του hat.

```
#GetSensorValues from sensorhat
def GetSensorValues(report):
    sense = SenseHat()
    report.timestamp = str(datetime.datetime.now())
    report.temperature = sense.get_temperature()
```

```
#mqtt callback functions
def on_log(client, userdata, level, buf):
    print("log:" + buf)

def on_connect(client, userdata, flags, rc):
    if rc==0:
        print("connection established")
    else:
        print("could not establish connection:", rc)

def on_disconnect(client, userdata, flags, rc=0):
    print("Disconnected with result code: " + str(rc))

def on_message(client, userdata, msg):
    actuation = pickle.loads(msg.payload)
    #Checking if the actuation is targeted for this device
    global currState
    if actuation.isOn != currState and actuation.targetId == "1":
        currState = actuation.isOn
        if str(actuation.isOn) == "1":
            sense.set_pixels(actuationScreen)
            print("LIGHTS ON BABY")
        else:
            sense.clear()
            print("lights off..")
```

```
#GetSensorValues from sensorhat
def GetSensorValues(report):
    sense = SenseHat()
    report.timestamp = str(datetime.datetime.now())
    report.temperature = sense.get_temperature()
    report.humidity = sense.get_humidity()
    report.pressure = sense.get_pressure()
    report.thingId = "1"
```

Μετά έχει τα MQTT callbacks -παρόμοια με το BotCode-. Το `pickle.loads` πρακτικά είναι η γραμμή που κάνει `deserialize` το μήνυμα που λαμβάνει. Το `PiCode` το μόνο `topic` που θέλει να ακούει είναι το `"discord/actuactions"` για το λόγο αυτό ελέγχει αν το `actuation.targetId` είναι 1 (υποτίθεται ότι το `unique ID` του συγκεκριμένου `thing` είναι το 1) και έτσι ξέρει π.χ. ότι το `llights` ο αναφέρεται σε αυτό.

```
#SenseHat displays
global X, I, 0, actuationScreen, sense
sense = SenseHat()
X = [255, 0, 0]
I = [0, 255, 0]
0 = [0, 0, 255]
actuationScreen = [
I, 0, 0, X, X, 0, 0, I,
I, 0, X, 0, 0, X, 0, I,
I, 0, 0, 0, 0, X, 0, I,
I, 0, 0, 0, X, 0, 0, I,
I, 0, 0, X, 0, 0, 0, I,
I, 0, 0, X, 0, 0, 0, I,
I, 0, 0, 0, 0, 0, 0, I,
I, 0, 0, X, 0, 0, 0, I
]
```

Αυτό το κομμάτι κώδικα είναι αποκλειστικά για το πώς θα ανάψει το led screen. Οι τιμές στα X,I,O είναι τριπλέτες RGB και όπως φαίνεται βγάζει ένα κόκκινο ερωτηματικό στη μέση με 2 πράσινες κατακόρυφες γραμμές σε μπλε φόντο.

Μετά κανονικά συνδέεται στον broker, κάνει τα MQTT callback function bindings και απευθείας δημιουργεί ένα ThingRegistration το κάνει pickle.dumps -δηλαδή serialize- και το κάνει publish στο "discord/thingRegistration" topic ώστε να κάνει register στο bot που έχει κάνει subscribe στο ίδιο topic. Κάνει subscribe στο "discord/actuations" ώστε να λαμβάνει τα actuations.

Τέλος μπαίνει σε ένα ατέρμον βρόγχο όπου κάθε 10 sec στέλνει ένα report στο "discord/inpt".

```
piClient.py
broker_addr = "iot.eclipse.org"
client = mqtt.Client("IoTClient")
#bindings
client.on_connect = on_connect
client.on_disconnect = on_disconnect
#client.on_log = on_log
client.on_message = on_message
#establishing server connection
print("connecting to broker...",broker_addr)
client.connect(broker_addr)
time.sleep(1)
print("connected!")

#Registration to discord bot client
print("making registration to the bot client...")
reg = ThingRegistration("1",["Temperature", "Humidity", "Pressure"])
payload = pickle.dumps(reg)
client.publish("discord/thingRegistration", payload, 0)
time.sleep(1)
print("registration done!")

#IoT thing routine start
report = Report(str(datetime.datetime.now()),10,10,10,"1")
GetSensorValues(report)
client.loop_start()
client.subscribe("discord/actuations")
time.sleep(1)

while 1:
    payload = pickle.dumps(report)
    client.publish("discord/inpt", payload, 0, True)
    GetSensorValues(report)
    time.sleep(10)
    client.disconnect()
    client.loop_stop()
```

Τελικά σχόλια:

Στο παραδοτέο έχω βάλει και τους 2 κώδικες αν και γνωρίζω ότι το bot μου ελέγχεται μόνο από την IP του σπιτιού μου ή του laptop άρα τους κώδικες μπορώ να τους τρέξω μόνο εγώ γι'αυτό είχα κάνει και το live demonstration στη παρουσίαση.

Ευχαριστώ!