Patryk Stroński 242564

Lab 6 – extension of Lab5

Introduction

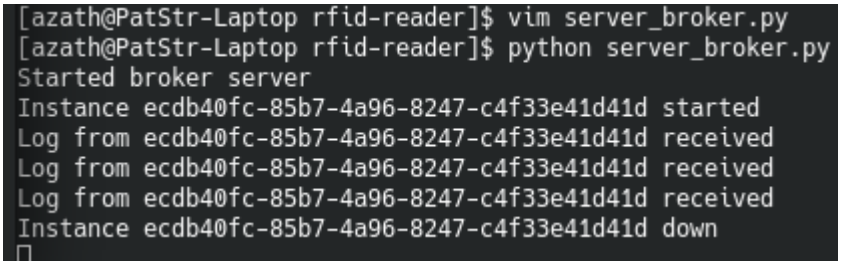
Lab 6 is an extension of lab 5, therefore the instruction from lab 5 is more or less valid (which is also in this zip). Even though there are several changes. First of all there are now three modules, where two of them need to run on server and one on RFID card reader RaspBerry PI. The *reader.py* file is the pne that needs to run directly on Raspi, whereas *server_broker.py* and *index.py* – on server. *Index.py* manages the flask-based REST interface and *server_borker.py* is responsible for message subscription. The logs of cards are transferred from Reader parts to Server using messages via MQTT protocol. The server needs therefore to have MQTT broker Mosquitto running. It can be easily downloaded either from repositories (like in case of Arch Linux – AUR), or installed as a package via snap. Please refer to the site https://mosquitto.org/download/. Then it needs to be run using command `mosquitto -c /etc/mosquitto/mosquitto.conf`. Where *etc/mosquitto/mosquitt.conf* is the filepath of the config file.
Each module is run from terminal using *python <filename.py>* command.
Also, as previously, remember to have these conditions satisified:
- postgresql database up and running with database *rfid_logs*
- port 5000 free and open
- credentials in *db_management* set up correctly.
- correct Mosquitto url in both *server_broker.py* and *reader.py* defined using *QUEUE_ADDRESS* variable
- the modules from *dependencies.txt* file installed using pip.
- the table card_logs needs to be dropped and the one from previous version of the exercise cannot persist. It changes in this exercise so it's better for it to self load using the creation statements in *db_management.py* file.

Message structure

The messages include json formatted (but stringified) message. The following Fields are there for log reading: *{"status": log, "time":* <time of reading card>*, "uid":* <employee card uid>*, "instance_id":* <uid of the worker (reader)>*}*
and for healthcheck:
{"status": "health", "on": 0/1*, "instance_id":* <uid of the worker (reader)>*}*
They are decoded by the broker module and processed.

Healthchecks

Healthchecks are special logs informing that an instance connected or disconnected to the broker. When the broker is working and reader is initialized, the log is printed by broker, informing that some reader is up and running. The uuid of this reader (instance id, not hardware reader id, because each running reader.py can for now use only one HW RFID reader) is provided. When the instance is connecting and is disconnecting, the message is sent. Healthchecks are implemented for logging purposes. Based on healthcheks from logs administrator can now that the reader got connected or not.



RFID reading

The main part is the RFID reading. The reads of the external card are sent using the MQTT protocol. The messages are saved immediately to the database and are browsable from table card_logs in postgresql. I ran a 2-second-gap to show that the logs are read and written to the database with correct data received from broker.

```
[azath@PatStr-Laptop rfid-reader]$ python server_broker.py
Started broker server
Instance 28b093b4-acfb-499e-87fa-bba52bdbb81b started
Log from 28b093b4-acfb-499e-87fa-bba52bdbb81b received
Log from 28b093b4-acfb-499e-87fa-bba52bdbb81b received
Log from 28b093b4-acfb-499e-87fa-bba52bdbb81b received
Log from 28b093b4-acfb-499e-87fa-bba52bdbb81b received
Log from 28b093b4-acfb-499e-87fa-bba52bdbb81b received
Instance 28b093b4-acfb-499e-87fa-bba52bdbb81b down
^CTraceback (most recent call last):
  File "server_broker.py", line 29, in <module>
    time.sleep(1)
KeyboardInterrupt

[azath@PatStr-Laptop rfid-reader]$ psql -d rfid_logs
psql (12.2)
Type "help" for help.

rfid_logs=# \d
                List of relations
 Schema |       Name       |   Type   |  Owner
--------+------------------+----------+----------
 public | card_logs        | table    | postgres
 public | card_logs_id_seq | sequence | postgres
 public | users            | table    | azath
 public | users_id_seq     | sequence | azath
(4 rows)

rfid_logs=# SELECT * FROM card_logs
rfid_logs-# ;
 id |   uid    |  log_time  |              reader_id
----+----------+------------+--------------------------------------
  1 | uid-test | 1587710813 | 28b093b4-acfb-499e-87fa-bba52bdbb81b
  2 | uid-test | 1587710815 | 28b093b4-acfb-499e-87fa-bba52bdbb81b
  3 | uid-test | 1587710817 | 28b093b4-acfb-499e-87fa-bba52bdbb81b
  4 | uid-test | 1587710819 | 28b093b4-acfb-499e-87fa-bba52bdbb81b
  5 | uid-test | 1587710821 | 28b093b4-acfb-499e-87fa-bba52bdbb81b
(5 rows)

rfid_logs=#
```

RESTful UI

The RESTful api skeleton that was introduced in the previous exercise remains unchanged. Please refer to the report from there. I only added *logs* property where I'm listing all logs from that day for an employee, where *in* and *out* have the first and last log of the given day.

Important note: you need to assign user yourself via API call to the UID, because in the beginning the UIDs can be read, but the user needs to have them assigned in order to work correctly.

As a result, the following report can be evaluated: