POLITECNICO DI TORINO

Cloud specifications

"Projects and laboratory on communication systems " class

Academic year 2018/19

Guido Albertengo, German Sviridov, Senay Semu Tadesse

Version: May 24, 2019

IMPORTANT NOTE: This documentation continuously undergoes modifications and is subject to change over time. If you have any issue related to the functionalities discussed here please send your inquiries to german.sviridov@polito.it.

Chapter 1

Database specification

An SQL database based on Firebird has been deployed for the purpose of this course.
The database is composed of 4 tables:

- customer: storing the information related to the customers
- logs: storing logs related ot messages published over MQTT
- configuration: stores status and information related to the devices.
- extension: stores any additional hardware extension related to the devices

1.0.1 Customer table

The customers is composed of the following fields:

- id: primary key, auto-increment
- group_id: equivalent to the username of a customer. By default each group is assigned a group_id "PL19-N" where N is the number of the Raspberry Pi (RP) given to each group. Note that N is represented over two digits (groups having a RP with $N < 10$ will have a group_id equal to "PL19-0N")
- group_psw: the password that will be used by each group to perform authentication to the services offered by the cloud. Passwords have been already preset and will be published alongside with this document.
- group_info: Any additional information related to the group. Each group is required to fill it **at least** with names, surnames and student ids of its members.

1.0.2 Logs table

- device_id: id of the device that generated the log entry. Foreign key is `configuration.id`
- timestamp_srv: the timestamp at which the log message has been processed. Supported format is `%Y-%m-%d %H:%M:%S.%f`.
- timestamp_dev: the timestamp at which the log message has been generated by the device.

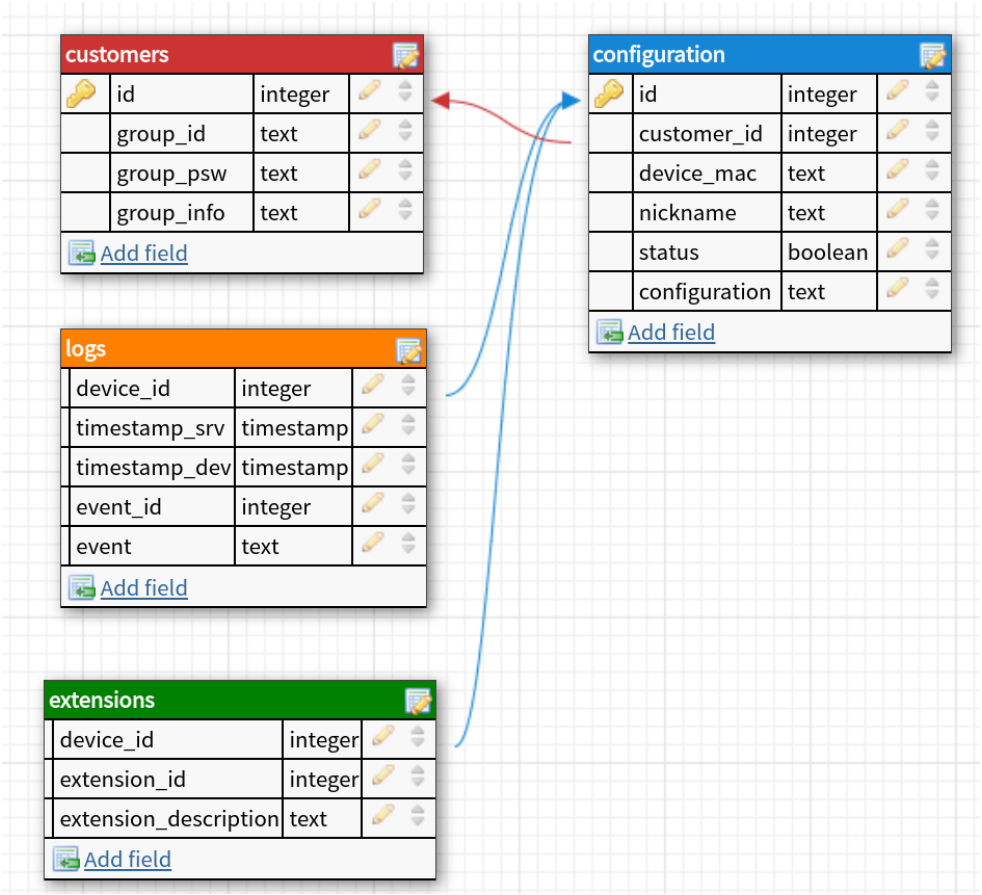

Figure 1.1: Database schema

- event_id: identification number of the generated event. See subsequent sections for a lsit of possible ids.
- event: text field containing information related to the generated event.

1.0.3 Configuration table

- id: primary key, auto-increment
- customer_id: foreign key which specifies the owner of the device.
- device_mac: the MAC address of the device
- nickname: textual identifier of the device given by the customer.
- status: binary value specifying whether the device is active or not. This field is employed by the ping service explained later in Sec. 2
- configuration: arbitrary string used to store device-specific configuration

1.0.4 Extension table

Left for further implementation

Chapter 2

MQTT specification

2.1 Connecting to the cloud broker

The broken is running on AWS and can be reached via MQTT at:
`a3cezb6rg1vyed-ats.iot.us-west-2.amazonaws.com`
Inside MQTT directory you can find example application of a client alongside with the security certificates. Similarly you can find an example of the server application which generates ping requests.

2.2 Available topics and their functionality

pl19/event

The event topic is used in order to communicate with the broker. Client are required to publish all updates to this topic.

pl19/notification

The notification topic is used by the server as a broadcast channel in order to publish information related to all customers.
Note: The server will not record anything you publish on this topic unless you register your device mac first! See Sec. 3.1.3 for details on how to do it.

2.3 Message structure

All published messages/events must include the following fields:

1. `event_id`: an integer value specifying the id of the generated event.
2. `timestamp`: the timestamp specifying the message generation time. See section 1 for details of the format.
3. `device_mac`: a string field containing the MAC address of the device which publishes the message. MAC address ”00:00:00:00:00:00” is reserved to the broker.
4. `event`: a string field containing arbitrary information about the event.

2.4 MQTT services

Ping service

The server will periodically publish a ping request on the notification channel. A ping request message can be identified by the following information:

1. `event_id` = 0
2. `event_data`: a JSON formatted string containing at least the following fields: {sequence:sequence, message:message}.

All clients are required to reply on the event channel by specifying an `event_id` = 1 and with an `event` field containing a JSON formatted string with the sequence number related to the ping request.

2.5 Lambda execution

A special Lambda function is executed on AWS whenever a new message is published on either of the channels. By default the Lambda function stores all messages inside the log table of the database. If the processed message is a ping reply the Lambda function updates the status of the device which generated the response by setting the `status` field of the transmitting device to 1 inside the `configuration` table. Statuses are instead reset to 0 whenever the lambda function processes a ping request message.

Chapter 3

Web service specification

The APIs are reachable at:
`http://ec2-34-220-162-82.us-west-2.compute.amazonaws.com:5002`
All APIs are written with Flask and treat all data as JSON strings.

3.1 Available APIs

3.1.1 Authentication

In order to use the APIs you are required an authentication token. The authentication token can be obtained via a call to the /**auth** API as follows:

/auth

Synopsis:	POST `/auth`
	/auth
Parameters:	{"username":username, "password":password}
Header:	Content-Type: application/json
Status code:	200 OK, 400 Bad Request

Response

Content-Type	application/json
Value	{"access_token":token}

example

```
curl -H "Content-Type: application/json" -X POST --data '{"username":"PL19-XX", "password":"XXXXX"}' localhost:5002/auth
{"access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZGVudGl0eSI6MTMwMSwiaWF0IjoxNTU3OTM3MTM3LCJuYmYiOjE1NTc5MzcxMzcsImV4cCI6MTU1NzkzNzQzN30.IujndzRXBksYASiYTWgTb1Z2QYtfQjazHQMfzYTpC8o"}
```

The token must be securely stored on the local machine and used for subsequent calls to the APIs. The token expires after 24h (or in the event of a system shutdown), thus it requires to be periodically re-generated.

3.1.2 Group information retrieval/update

Groups can modify information related to their groups via the following API:

/user/::group_id

Synopsis:	GET /user/::group_id /user/::group_id
Parameters:	
Header:	Authorization: JWT ::token
Status code:	200 OK, 400 Bad Request

Response

Content-Type	application/json
Value	{"group_info": group_info, "group_id": group_id, "group_psw": group_psw}

example

```
curl -H "Authorization: JWT eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJpZGVudGl0eSI6MTMwMSwiaWF0IjoxNTU3OTM3Nzc5LCJuYmYiOjE1NTc5Mzc3NzksImV4cCI6MTU1NzkzODA3OX0
    .8GvnwspA6Ly6SufDVFunL9iVxfcYzZtbDz7R4eroPUM" localhost:5002/user/PL19-XX
"{\"group_info\": \"{}\", \"group_id\": \"PL19-XX\", \"group_psw\": \"XXXXX\"}"
```

Synopsis:	POST /user/::group_id /user/::group_id
Parameters:	{"group_info": group_info, "group_id": group_id}
Header:	Content-Type: application/json Authorization: JWT ::token
Status code:	200 OK, 400 Bad Request

Response

Content-Type	application/json
Value	

example

```
curl -H "Authorization: JWT eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJpZGVudGl0eSI6MTMwMSwiaWF0IjoxNTU3OTM4MTY3LCJuYmYiOjE1NTc5MzgxNjcsImV4cCI6MTU1NzkzODQ2N30
    .-0GRDYUfqRYYakoIOhcbH3KpC0vWw7K3swaeCTbdCAw" -H "Content-Type: application/json" -X POST
    --data '{"group_id": "PL19-XX", "group_info":"Test info", "configuration":"test config"}'
    localhost:5002/user/PL19-XX
"{\"message\": \"Group information updated\", \"code\": 200}"
```

3.1.3 Device information retrieval/update

Groups can modify information related to their groups via the following API:

/user/::group_id/devices

Synopsis:	GET /user/::group_id/devices /user/::group_id/devices
Parameters:	
Header:	Authorization: JWT ::token
Status code:	200 OK, 400 Bad Request

Response

Content-Type	application/json
Value	"{"status": status, "data": {"configuration": configuration, "nickname": nickname, "device_status": device_status, "device_mac": device_mac}}"

example

```
curl -H "Authorization: JWT eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJpZGVudGl0eSI6MTMwMSwiaWF0IjoxNTU3OTM5MzU0LCJuYmYiOjE1NTc5MzkzNTQsImV4cCI6MTU1NzkOTY1NH0
    .8RWyg-CYJykqaEbmoYZAb_0hPdp-cldcqRJOcKbjuWk" localhost:5002/user/PL49/devices
"{\"status\":200,\"data\":{\"nickname\":\"Dev\",\"configuration\":\"test config\",\"
    device_mac\":\"00:00:00:00:00:00\",\"device_status\":0}}"
```

Synopsis:	POST /user/::group_id/devices /user/::group_id/devices
Parameters:	{"device_mac":"device_mac, "nickname": nickname, "configuration": configuration}
Header:	Content-Type: application/json Authorization: JWT ::token
Status code:	200 OK, 400 Bad Request

Response

Content-Type	application/json
Value	

example

```
curl -H "Authorization: JWT eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJpZGVudGl0eSI6MTMwMSwiaWF0IjoxNTU3OTMyMjE4LCJuYmYiOjE1NTc5MzIyMTgsImV4cCI6MTU1NzkzMjUxOH0
    .J-zPA0o1dKyliHL_goiqhbKY2kSDSywS5aNV83AQu8c" -H "Content-Type: application/json" -X POST
    --data '{"device_mac":"AA:00:00:00:00:00","nickname":"Dev","configuration":"test config
    "}' localhost:5002/user/PL19-49/devices
```

3.1.4 Logs retrieval

Logs related to a particular device can be retrieved via a specific call. The parameter of the call is group id as it is assumed for simplicity to have a 1-to-1 mapping between a group and a device.

/user/::group_id/logs

Synopsis:	GET /user/::group_id/logs
	/user/::group_id/logs
Parameters:	
Header:	Authorization: JWT ::token
Status code:	200 OK, 400 Bad Request

Response

Content-Type	application/json
Value	"{"status": status, "data": [{"event_id": event_id, "timestamp_dev": timestamp_dev, "timestamp_srv": timestamp_srv, "event": "{"message": message, "sequence": sequence}", "device_id": device_id}, ...]}"

example

```
curl -H "Authorization: JWT eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJpZGVudGl0eSI6MTMwMiwiaWF0IjoxNTU3OTM1MTU2LCJuYmYiOjE1NTc5MzUxNTYsImV4cCI6MTU1NzkzNTQ1Nn0
    .C1l1j_vw47kPGR9t0lw3nM9Y56XRqSAJG8IvsKH56FE" localhost:5002/user/PL19-XX/logs
"{\"status\": 200, \"data\": [{\"event_id\": 0, \"timestamp_dev\": \"2019-05-15
    15:49:18.442800\", \"timestamp_srv\": \"2019-05-15 15:49:19.023500\", \"event\": \"{\\\"
    message\\\": \\\"Please reply.\\\", \\\"sequence\\\": 0}\", \"device_id\": 639}, {\"
    event_id\": 0, \"timestamp_dev\": \"2019-05-15 15:49:28.475500\", \"timestamp_srv\":
    \"2019-05-15 15:49:28.762400\", \"event\": \"{\\\"message\\\": \\\"Please reply.\\\", \\\"
    sequence\\\": 1}\", \"device_id\": 639}]}"
```

3.2 Website functionalities

A frontend service is available at

`http://ec2-34-220-162-82.us-west-2.compute.amazonaws.com`

You can login to the website with your credentials. Once inside you can modify group information such as password and group information. The website exploits previously discussed APIs to communicate with the backend.