UNIVERSITÀ DEGLI STUDI
DI GENOVA

Department of Computer Science

IoT - Internet of Things A.Y. 2019/2020

Unige Crowd Detector API

Tiziano Firpo (S 4379124)
Andrea Canepa (S 4185248)

August 2020

Contents

1 Notes **2**

2 Buildings endpoints **3**
2.1 `/registerNewBuilding` 3
2.2 `/getBuildings` 4
2.3 `/deleteBuilding` 5

3 Sensors endpoints **6**
3.1 `/registerNewNode` 6
3.2 `/getNodes` 7
3.3 `/deleteNode` 8

4 Data endpoints **9**
4.1 `/getSimpleStatistics` 9
4.2 `/updateCrowd` 10
4.3 `/getStatistics` 11

1 Notes

`Unige Crowd Detector` exposes a *REST API*, described in this document. `Unige Crowd Detector` *API* allows to interact with the server, getting or updating the number of people in a room along with a small set of statistics.

The end user could interact with the *API* using different **endpoints**. Each endpoint accepts a set of parameters and returns values formatted in `JSON`. Allowed methods to reach endpoints are: `GET` and `POST`. `POST` methods endpoints accepts `JSON` formatted payload.

Parameter and return types defined in this document follows some rules:

- ***types*** that end with **?** define that the corresponding value is not mandatory.
- ***types*** that end with + stand for values that could occur multiple times.

All the other parameters are mandatory and in case of multiple definitions an error code will be returned.

2 Buildings endpoints

2.1 `/registerNewBuilding`

By calling this endpoint a new building will be registered in the server. It's recommended to give a meaningful `name` *to the building.*

- **METHOD:** POST
- **PARAMETERS:**
  1. **name:** ***string*** = *name* of the building, for example "DIBRIS"
  2. **address:** ***string?*** = *address* of the building
  3. **numFloors:** ***int*** = how many *floors* are in the building (for simplicity we assume that buildings have floors in range [1, numFloors])
  4. **token:** ***string*** = to identify the *administrator*
- **RETURNS:**
  1. **code:** ***int*** = define if the *operation* was successful or which type of *error* was encountered
     - **42**: OK
     - **442**: generic error
     - **443**: bad data received
     - **444**: authentication fail
  2. **id:** ***string?*** = *private identifier* of the *building*, it will be used to identify a building to interact with this API

2.2 `/getBuildings`

By calling this endpoint you retrieve information about one or more building registered in the server. If `name` *is not provided all buildings are returned.*

- **METHOD:** GET
- **PARAMETERS:**
  1. **name:** ***string?*** = name of the building, for example "DIBRIS"
- **RETURNS:**
  1. **code:** ***int*** = define if the *operation* was successful or which type of *error* was encountered
     - **42**: OK
     - **442**: generic error
     - **443**: bad data received
  2. **buildings:** ***JSON array*** = each object represents a building

$$\begin{cases} \textbf{name: } \textbf{\textit{string}} = \text{name of the building,} \\ \textbf{address: } \textbf{\textit{string}} = \text{address of the building,} \\ \textbf{numFloors: } \textbf{\textit{int}} = \text{number of floors in the building} \end{cases}$$

2.3 `/deleteBuilding`

By calling this endpoint the administrator can remove the building from the server. After this operation, the name belonging to the deleted building will be available again.

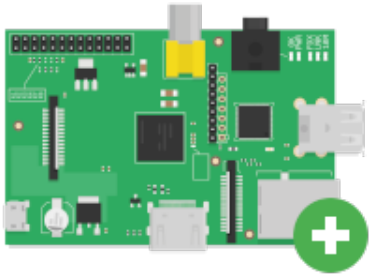
- **METHOD:** POST
- **PARAMETERS:**
  1. **name:** ***string?*** = name of the building to remove, for example "DIBRIS"
  2. **token:** ***string*** = secret of the admin
- **RETURNS:**
  1. **code:** ***int*** = define if the *operation* was successful or which type of *error* was encountered
     - **42**: OK
     - **442**: generic error
     - **443**: bad data received
     - **444**: authentication fail

3 Sensors endpoints

3.1 `/registerNewNode`

By calling this endpoint a new node will be registered in the server, it's suggested to give a meaningful name to the node (i.e. the area where it will operate). ***`name`*** *must be unique per* ***`floor`*** *and* ***`building`****.*

- **METHOD:** POST
- **PARAMETERS:**
  1. **name:** ***string*** = name of the node, for example the name of the area in which it will operate
  2. **max_people:** ***int*** = maximum number of people allowed in that room
  3. **type:** ***string*** = type of room
  4. **floor:** ***int*** = floor at which the room is (in the range of the building)
  5. **building:** ***string*** = name of the building
  6. **token:** ***string*** = to identify the administrator
- **RETURNS:**
  1. **code:** ***int*** = define if the *operation* was successful or which type of *error* was encountered
     - **42**: OK
     - **442**: generic error
     - **443**: bad data received
     - **444**: authentication fail
  2. **id:** ***string*** = private identifier of the node, it will be used to identify a node to interact with this API

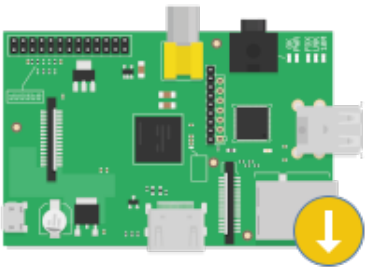


3.2 `/getNodes`

By calling this endpoint you can retrieve information about one or more sensors inside of a specific building.

- **METHOD:** GET
- **PARAMETERS:**
  1. **building:** ***string*** = building from which you want to retrieve the sensors
  2. **floor:** ***int?+*** = floor from which you want to retrieve the sensors
  3. **roomType:** ***string?+*** = type of room from which you want to retrieve the sensors
- **RETURNS:**
  1. **code:** ***int*** = define if the *operation* was successful or which type of *error* was encountered
     - **42**: OK
     - **442**: generic error
     - **443**: bad data received
  2. **listOfNodes:** ***JSON array*** = list of nodes with id, name, type and floor

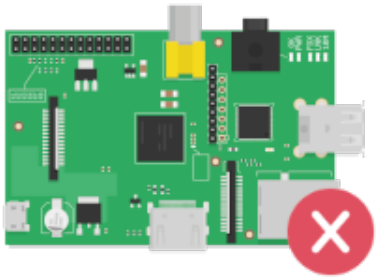
     { **id:** ***int*** = identifier of the node,
     name: ***string*** = name of the node,
     type: ***string*** = type of room where the node is placed,
     floor: ***int*** = floor of the node,
     maxPeople: ***int*** = the capacity of the room



3.3 `/deleteNode`

By calling this endpoint the administrator can remove a the node from the server. After this operation, the name belonging to the deleted node will be available again.

- **METHOD:** POST
- **PARAMETERS:**
  1. **id:** ***string*** = private identifier of the node (this id is provided by the server during the registration)
- **RETURNS:**
  1. **code:** ***int*** = define if the *operation* was successful or which type of *error* was encountered
     - **42**: OK
     - **442**: generic error
     - **443**: bad data received



4 Data endpoints

4.1 `/getSimpleStatistics`

By calling this endpoint you retrieve only the current count of people in rooms detected by the sensors. It is useful for the main page of the web application or to have a quick overview of the situation in the building.

- **METHOD:** GET
- **PARAMETERS:**
  1. **id:** ***string+*** = to see one or more specific areas covered by nodes present in the list, if not present it will return a list of all the lists of nodes
- **RETURNS:**
  1. **code:** ***int*** = define if the *operation* was successful or which type of *error* was encountered
     - **42**: OK
     - **442**: generic error
     - **443**: bad data received
  2. **datas:** ***JSON array*** = each object has the following shape:

     $$\begin{cases} \textbf{id: } \textbf{\textit{int}} = \text{identifier of the node,} \\ \textbf{num: } \textbf{\textit{int}} = \text{number of people at the moment in that room} \end{cases}$$

4.2 `/updateCrowd`

By calling this endpoint the sensor update the count of people in the area in which is placed on the server. The detection is implemented through Bluetooth.

- **METHOD:** POST
- **PARAMETERS:**
  1. **id:** ***string*** = private identifier of the node
  2. **current:** ***int*** = number of current people present in the area
  3. **new:** ***int*** = number of new people since the beginning of the day
  4. **time:** ***UTC timestamp*** = time of detection of the people in the room
- **RETURNS:**
  1. **code:** ***int*** = define if the *operation* was successful or which type of *error* was encountered
     - **42**: OK
     - **442**: generic error
     - **443**: bad data received

4.3 `/getStatistics`

By calling this endpoint you retrieve statistical data about a sensor in a building within a certain time range. You can perform different operations. It is useful for the web application since it returns information which are plottable.

- **METHOD:** GET
- **PARAMETERS:**
  1. **id:** ***string***+ = to see one or more specific areas covered by nodes present in the list, if not present it will return a list of all the lists of nodes
  2. **op:** ***string*** = to determine which operation to perform on data, one among `avg|max|distinct|all`
  3. **optionRange:** ***string*** = a value among `today|yesterday|lastweek|lastmonth`
- **RETURNS:**
  1. **code:** ***int*** = define if the *operation* was successful or which type of *error* was encountered
     - **42**: OK
     - **442**: generic error
     - **443**: bad data received
  2. **datas:** ***JSON array*** = each object has the following shape:

$$\begin{cases} \textbf{id: } \textbf{\textit{int}} = \text{identifier of the node,} \\ \textbf{datas: } \textbf{\textit{JSON array}} = \text{each value describe the number of people in a room (wrt. op) in an interval of time} \end{cases}$$