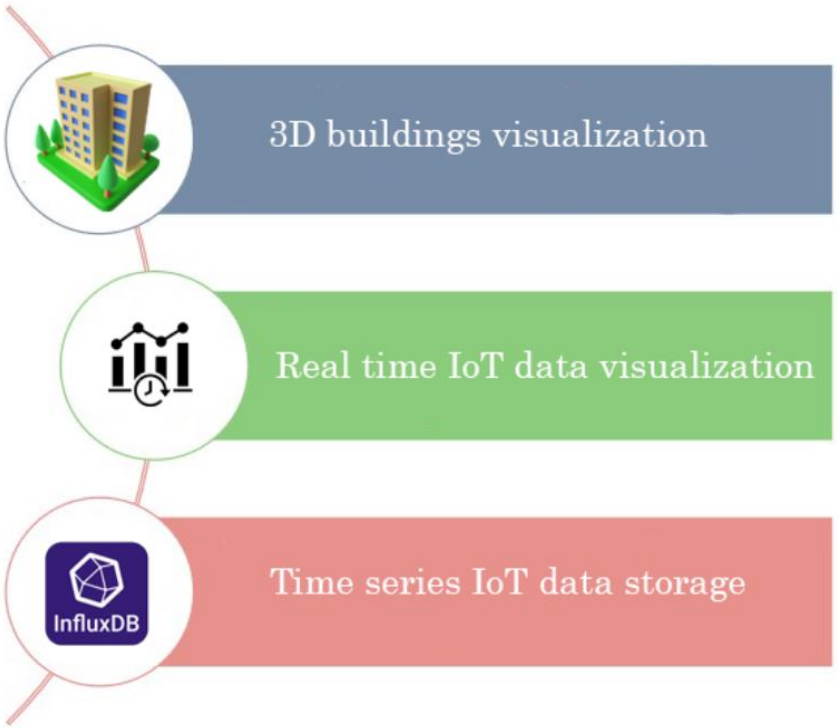
# IoT devices data visualization with Cesium

Torlini Matia s281431

# The idea : overview

- **Objective :**
  - **Develop an informative system for visualizing real time data and historical data coming from IoT devices within the Cesium 3D viewer**

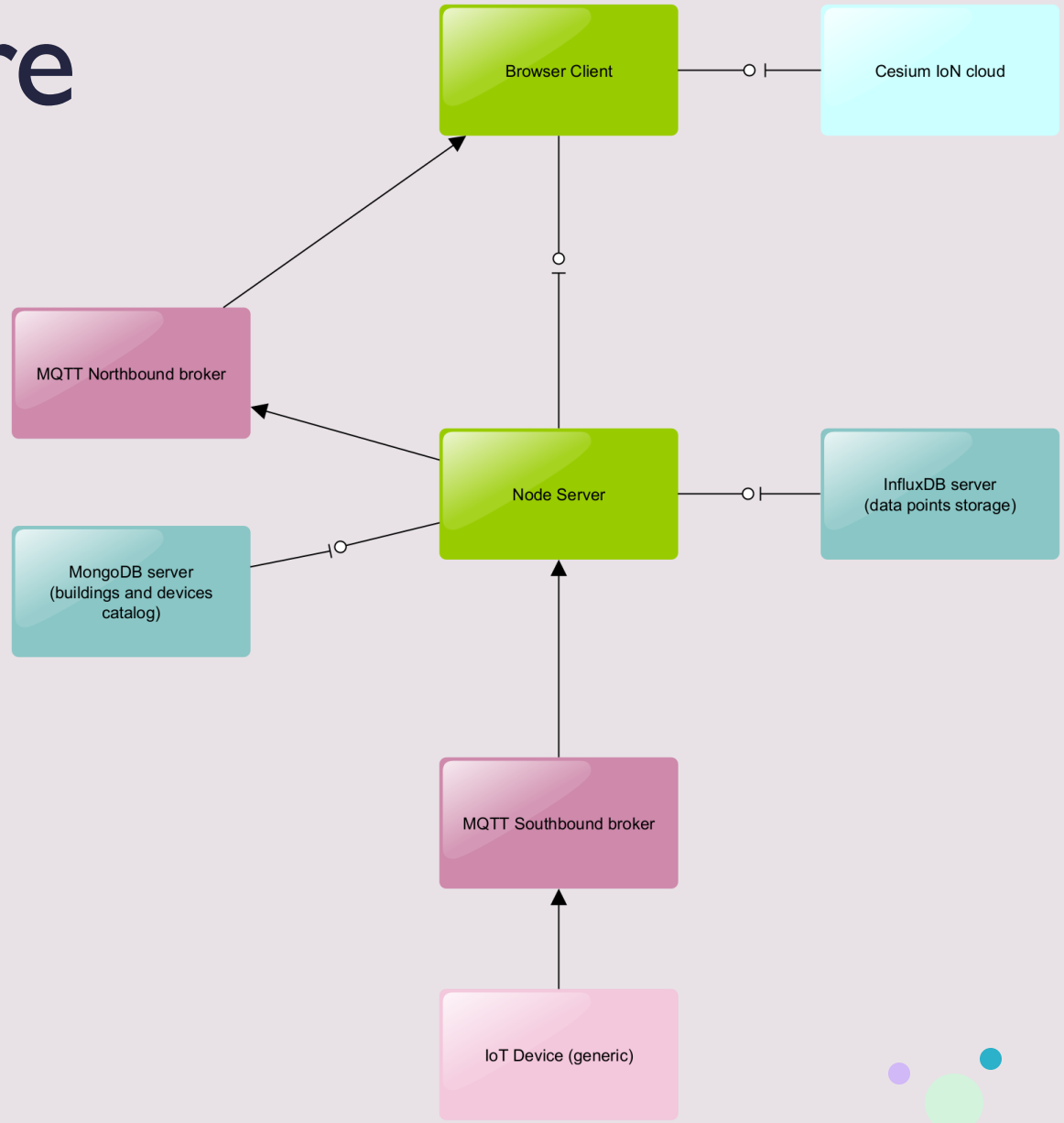


# Main architecture

MQTT communication

Rest service consumer

Rest service producer



# Communication protocols

- HTTP to manage entities in the architecture that follow the client-server paradigm :
  - Communications between web browser and Node server
  - Communications between Node server and MongoDB service
  - Communications between Node server and InfluxDB service
- MQTT to allow IoT devices to send their data in real time to entities in the architecture following the publish-subscribe paradigm :
  - Communication between IoT device and Node server
  - Communication between Node server and web browser client

# Cesium IoN cloud

- Cesium IoN is a Software as a Service (SaaS) which stores cityGML files with geometries and metadata of buildings as "assets"
- 3D tiles is the format used by the Cesium viewer to display 3D buildings
- Cesium cloud requires an account and an Access token

- To retrieve an asset from the Cesium IoN cloud, CesiumJS exposes a method

```
tileset =
await Cesium3DTileset.fromIonAssetId(tileset_id)
```

# MongoDB buildings collection document structure


```
{
 "_id": {
  "$oid": "65b0f89aa5ae79be64013fbb"
 },
 "CityObjectID": "BLDG_1140001",
 "Devices": [
  {
   "device___id": "123",
   "device___name": "Awesome device",
   "device___description": "The most awesome device",
   "device___status": "active",
   "fields": [
    {
     "name": "temperature",
     "value": ""
    },
    {
     "name": "humidity",
     "value": ""
    }
   ]
  }
 ]
}
```


- CityObjectID : it is the ID which identifies a building within the cityGML file
- Devices : an array of devices currently registered on the building
- Fields : an array of quantities the device publishes

# MQTT device topic and payload

- The MQTT message published by IoT devices
  are in the following format
- Here an example payload

```
Payload =
 {
  "value" : 10.1,
  "unit" : "°C",
  }
```

- The semantics about values in the payload is embedded into the topic on which data are published; topics are in the following format

  City/Building_id/Device_id/field

- As an example, the payload on the left is published on

  City/BLDG_1140001/123/temperature

# InfluxDB

- InfluxDB uses Line Protocol (LP) to insert data into buckets (databases)
- Line protocol is a data format with the following syntax

  <measurement>[,<tag_key>=<tag_value>[,<tag_key>=<tag_value>]]
  <field_key>=<field_value>[,<field_key>=<field_value>] [<timestamp>]

- As an example, the payload described before when inserted into influx db has the following format :

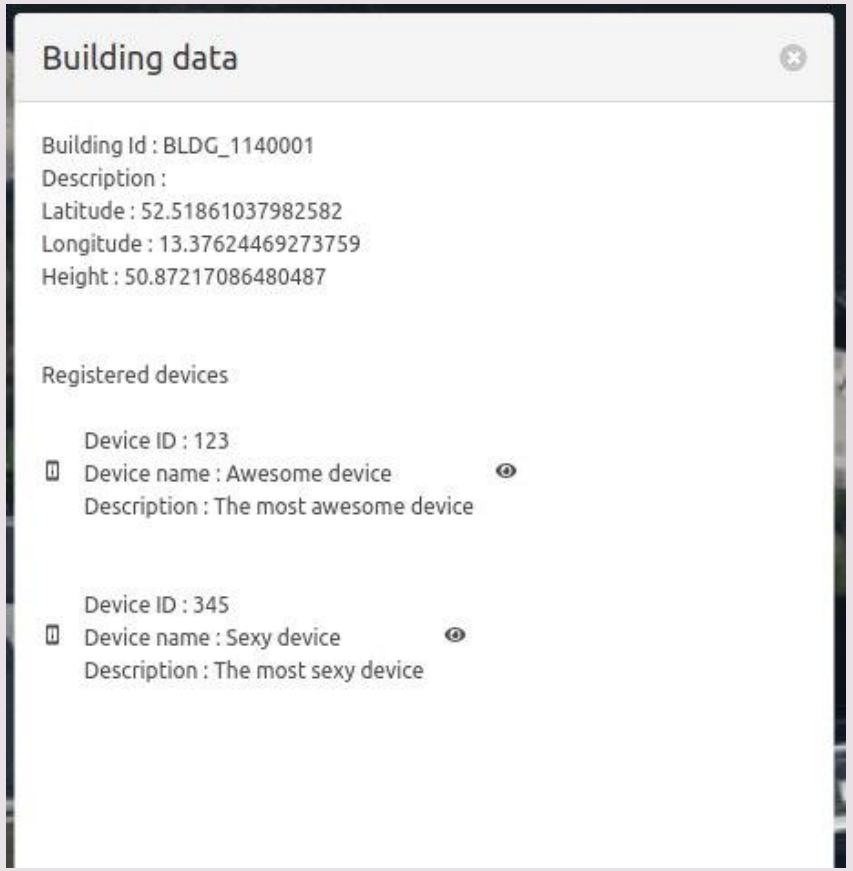
```
id_run_1, device_id="123", unit="°C", temperature=10.1  1556813561098000000
```

# Node Server

- Server's tasks :
  - (1) Collect IoT data and save them into influxDB
  - (2) Forwards received mqtt messages from the southbound broker to the northbound broker
  - (3) Interact with the MongoDB to access specific building data when requested by the browser client
  - (4) Interact with the InfluxDB service to retrieve historical data when requested by the browser client
- Server's MQTT subscriptions :
  - (1) subscribes to City/# at the southbound broker
- Server's MQTT pubblications :
  - (2) publishes on the same topic on which messages are received, but to the northbound broker
- Server's HTTP-API :
  - (3) POST  host:port/api/devices
  - (4) POST  host:port/api/measurements

# Browser client overview

- Browser client is written in React Javascript and styled with the Bulma css framework
- It makes use of the CesiumJS library to display with the Cesium Viewer 3D tiles hosted on the Cesium IoN cloud
- Each building is a clickable entity : when clicked, an HTTP-POST to server:port/api/devices is sent, with the building_id of the building entity in the body
- Response data are used to render the list of devices registered in the building, as well as buttons used to identify the fields of the devices we are interested in (field-buttons)



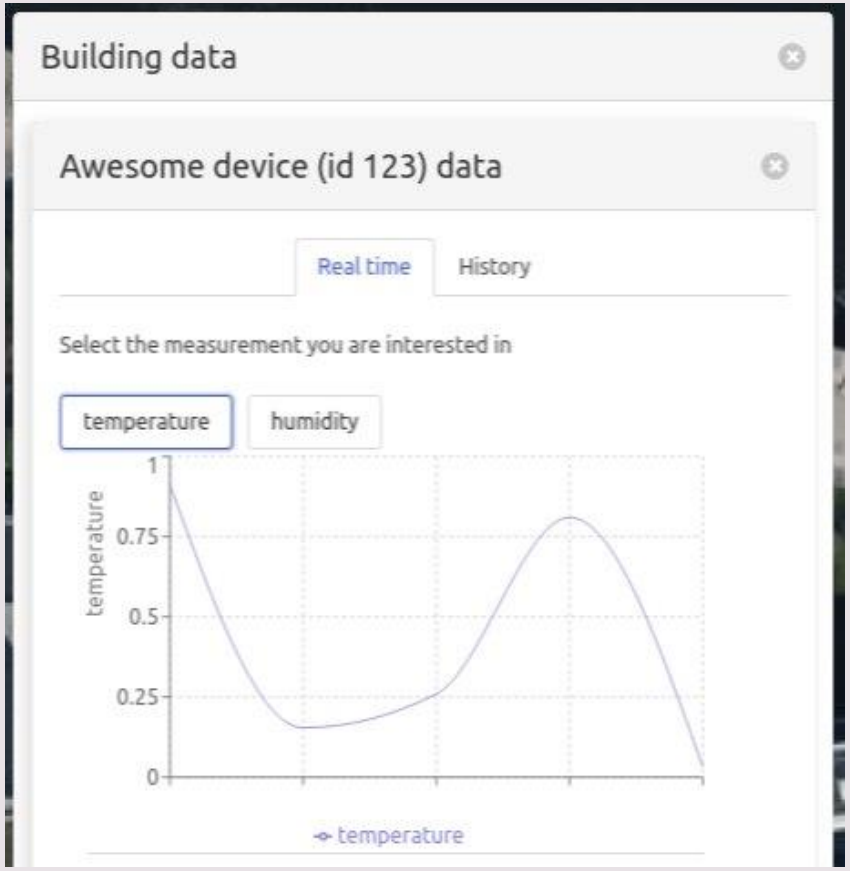
# Browser client - real time tab

Realtime data visualization tab:

whenever a field-button is pressed, the client subscribes to the

City/building_id/device_id/field

topic to receive real time data from the device

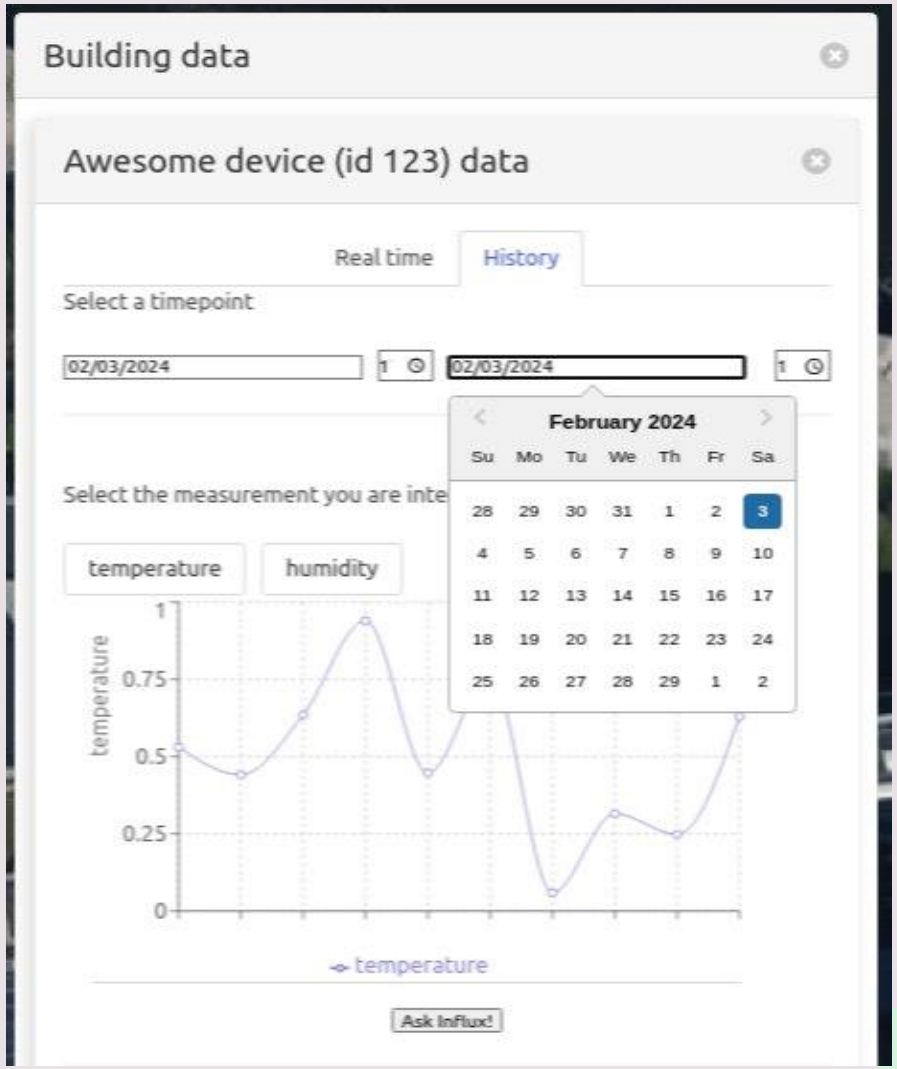


# Browser client - history tab

The user can set the data field he is interested in (pressing a field-button) and the time range he is interested in to see the trend of published field values within the selected time window.

The "Ask influx!" Button triggers the fetch of a HTTP – POST to

server:port/api/measurements



# The end