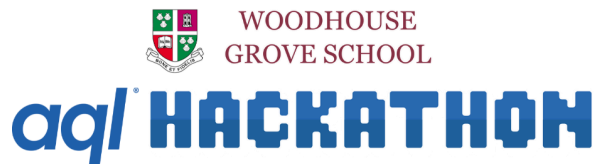




Woodhouse Grove Hackathon



Introduction

Hi, Andrew here. Welcome to our Smart Cities Hackathon! Today, you will learn about smart cities, 5G, IoT, LoRaWAN, coding using Node.js, Javascript, HTML, CSS, HTTP, Docker, Python data visualisation and analysis, and more.

All of these acronyms and abbreviations may sound intimidating, but don't worry; these are just various types of connectivity and the names of a few different coding languages. If you get stuck, there will be an AI assistant on hand to help you. Your assistant can walk you through each step of the challenges and possesses a deep knowledge of core.aql.com and the API documentation for api.core.aql.com. Since AI is now a normal part of our day-to-day lives, we see this as a key tool for the future of development. This is why we wanted to incorporate it into today's event. Your AI assistant has all the links to the tech you will use across the event, all of which are free to use. You will have access to thousands of sensors from all of our real-world 5G IoT testbeds, including data from the Eden Project, Mobile Access North Yorkshire, and other managed aql locations.

Have an amazing hack!

Andrew Morris,

VP Product & Development, aql

Who is aql?

About aql

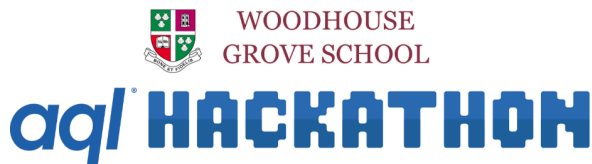
Founded in 1998 by Professor Adam Beaumont, aql is a trusted communications provider specialising in secure and reliable wholesale access to fixed and mobile networks for government, enterprise, and innovators. Our mission is to power a connected society by providing robust and novel solutions that enable communications between cities, communities, people, and businesses. We strive to make communication easy and accessible for all organisations to embrace the challenges a new digital world creates.

Our Role in IoT and 5G

As one of the UK's most innovative telecoms operators, aql has been at the forefront of developing and deploying 5G and IoT technologies. We have launched several 5G and IoT testbeds, including the Eden Project, Connected Cowes, Factory of the Future, and Mobile Access North Yorkshire. These initiatives demonstrate our commitment to driving technological innovation and providing real-world applications of IoT and 5G connectivity.

Why We Do What We Do

At aql, we believe in the transformative power of technology to drive positive change. Our vision is to support a smarter society that lives in closer harmony with our environment. By using our Core IoT platform and 5G mobile network, we enable innovators to explore the next generation of real-time technologies and applications. This innovation unlocks a broad



range of opportunities, from driving better decision-making to creating outstanding user experiences.

Who from aql is attending the hackathon?

Greg Monk - Marketing Executive

Greg is the marketing executive at aql. He is responsible for the company's written and visual marketing materials, site content and copy, event promotion and management and social media.

Spencer Maclean - Product Manager

Spencer has spent the last 7 years working in mobile app development, e-commerce, telecoms and wholesale aggregator solutions. As aql's Product Manager, he's responsible for product innovation, delivering good customer experience and more recently, using AI to share knowledge. After work, Spencer enjoys CrossFit and long walks with his family and husky.

What is aql's Core IoT platform?

aql's Core IoT platform is designed to provide a comprehensive solution for IoT applications that is sensor agnostic, meaning it can be used with multiple different kinds of sensors. Our platform is built to integrate seamlessly with various sensor technologies, including LoRaWAN, low-power Bluetooth, and serial devices, enabling users to manage and analyse sensor data efficiently and effectively.

Key Features

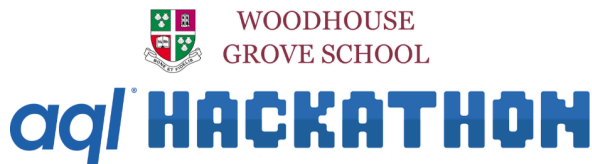
End-Connectivity Agnostic: The platform supports multiple communication technologies, ensuring flexibility and ease of integration with various sensor types without being locked in to any one vendor.

Real-Time Data Transformation: It transforms raw sensor readings into readable insights, making data easy to understand and use.

API-First Approach: Complete control over IoT sensor and device management through our API, supporting MQTT, LoRaWAN, and HTTP protocols.

Mobile Responsive: The platform is fully accessible on mobile devices, providing 24x7 reporting capabilities, real-time charting, and smart alarms.

Custom Dashboards: Users can create bespoke monitoring dashboards, shareable across organisations, to visualise and manage data effectively.



Secure and Scalable: The platform ensures secure communications and scalability from single to multiple locations with hundreds of sensors.

Integration Capabilities: Supports integrations with popular tools like InfluxDB, Grafana, Zabbix, Slack, and more, as well as aql's telephony services including SMS messaging and text-to-speech.

Hackathon Challenge Overview

Challenge 1: Create a virtual sensor for the other challenges

With this challenge you will create a virtual sensor of your choice within aql's Core IoT platform, this could be a temperature sensor, an air quality sensor, or a vibration sensor. If you need an idea you can ask the AI assistant. Once created we will use the Virtual sensor in other challenges including building a web front end and a Python script to generate random readings for your sensor. You will learn how to send a reading using the Core IoT API.

Challenge 2: Create a custom IoT chart

Our Core IoT platform allows you to view data from any sensor within the platform. Within this challenge, you will select a sensor from one of aql's IoT testbeds and will create a custom IoT chart that you can share with family and friends after the event.

Challenge 3: Create custom IoT analytic widgets

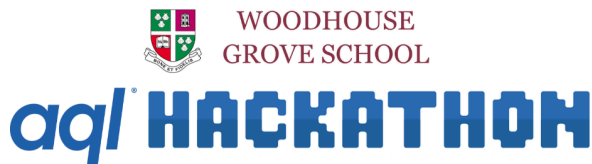
Our Core IoT platform allows you to create custom dashboards and analytics including line, bar, and scatter charts. You can create widgets that combine data over any period. Once the widget has been built, you can design your own dashboard. With this challenge, you will create a new Core IoT analytics widget and report and add this to your hackathon team location dashboard, as well as creating a unique dashboard that you will use for testing across the other challenges.

Challenge 4: Create a live Twitter Integration

Our Core IoT platform allows you to create social media posts for live IoT sensor readings. With this challenge, you will be able to create a new Twitter/X integration with a sensor of your choosing.

Challenge 5: Use Python to read a sensor's latest reading.

Our Core IoT platform API provides the ability to control all aspects of the platform including retrieving data from a single sensor or a selection of sensors. With this challenge, you will



create a Python script to query the latest value from a selection of sensors and will display these to the console.

Challenge 6: Extend your virtual sensor to generate random sensor readings using Python.

Our Core IoT platform allows sensor readings to be uploaded over a selection of protocols, such as LoRaWAN, HTTP, and MQTT. With this challenge, you will extend the virtual sensor you built in challenge 1 and you will create a Python script that will generate random sensor readings and upload these securely over HTTP using the Core IoT API. This Python code will run for the duration of the hackathon, generating random readings that will be used for the remaining challenges.

Challenge 7: Create a custom web page to display live sensor data.

In the earlier challenges, you will have created a virtual sensor and a Python script to generate live readings for your virtual sensor. With this challenge, you will create a web page using HTML, CSS, and Javascript to capture readings from your virtual sensor and plot your sensor readings in a line graph. If you're feeling daring, you can extend this challenge and get as creative as you like with it.

Challenge 8: Create widgets for your sensors.

In challenge 3 you created your first widget using an aql sensor from our 5G IoT testbeds. With this challenge we will create more widgets for each of the sensors you created within your virtual sensor, can you create a bar, line and scatter chart and create your own custom hackathon dashboard to demonstrate all the sensors you have used across the hackathon

Hackathon Challenge Detail

Challenge 1: Create a virtual sensor for the other challenges

This challenge will see you create a virtual sensor for core.aql.com. This will allow you to upload readings via the API to simulate real readings. We will then use this sensor for other challenges. This challenge will allow you to learn about <https://core.aql.com> from end to end, providing you with the knowledge and skills to complete and build upon in the other challenges. Good luck and enjoy.

Step 1

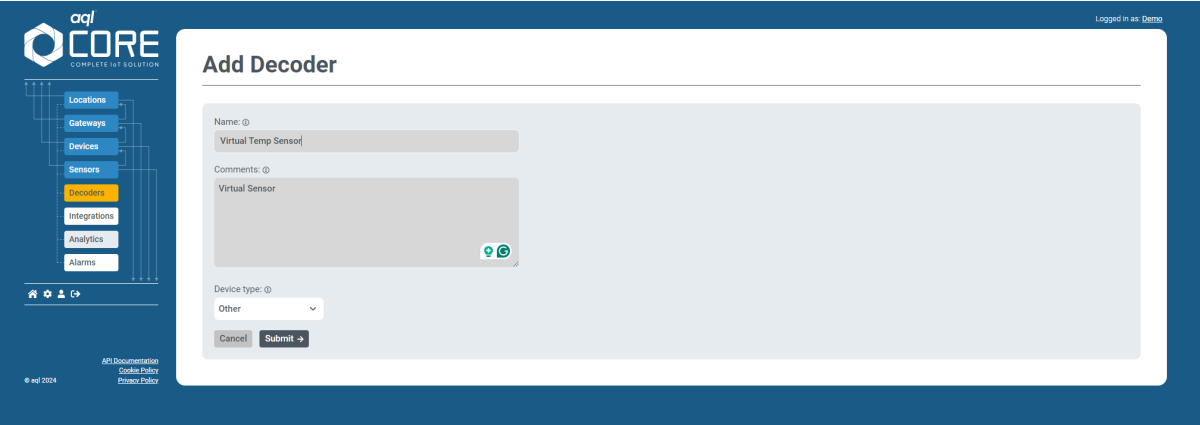
Log into the aql Core IoT platform (<https://core.aql.com/login>) with your provided event credentials.

Step 2

You will see a menu on the left-hand side of the screen.

- Select the 'Decoders' option. A new screen will appear, listing all decoders that have been created.
- For this challenge we are going to create a new decoder, so select the add Decoder from the top right part of the screen.
- The 'Add Decoder' screen will then appear. As each decoder must have a unique name, you can use the name of your team and a name for your new device. For example, Team1-Temp-Virtual Device.
- Enter a comment and set '**Device Type**' to '**Other**'.
- Once you are happy with the name of the decoder, press the 'Submit' button.

This step is now complete.



The screenshot shows the 'aql CORE' interface. On the left is a navigation menu with options: Locations, Gateways, Devices, Sensors, Decoders (highlighted), Integrations, Analytics, and Alarms. The main content area is titled 'Add Decoder'. It contains a form with the following fields: 'Name' with the value 'Virtual Temp Sensor', 'Comments' with the value 'Virtual Sensor', and 'Device type' set to 'Other'. At the bottom of the form are 'Cancel' and 'Submit' buttons. The top right corner of the interface indicates 'Logged in as Demo'.

Step 3



aqi HACKATHON

The decoder page allows you to create complex logic to decode sensor readings. Sensor readings are normally returned as a byte array (a collection of bytes). In this example, we are going to build a JSON object that acts as a template for the test decoder payload. This is a good task for the AI Assistant - you can ask it to create a JSON object for a virtual sensor. Below is an example of a suitable JSON object that you can use to get the AI started. Your virtual device can handle multiple sensor readings such as temperature, humidity, CO2.

```
{
  "timestamp": "2024-04-15T08:30:00Z",
  "location":{
    "latitude": 38.8951,
    "longitude":-77.0364
  },
  "Temperature": 25.5,
  "humidity": 50,
  "co2": 400,
  "air_quality":{
    "pm25": 10,
    "pm10": 20
  }
}
```

If you haven't built a JSON object before, the web page <https://jsoneditoronline.org/> can help you structure your class.

Once you have your JSON object, paste this into the 'Test Decoder' 'Payload' box.

Logged in as: Demo

AI Decoder

[Edit Decoder](#)

Comments:
AI Decoder

Device type: ⓪

Other

Decoder: ⓪

```
function parsePayload(payload, params) {
  return payload;
}
```

Test Decoder

Payload: ⓪

```
{
  "timestamp": "2024-04-15T08:30:00Z",
  "location": {
    "latitude": 38.8951,
    "longitude": -77.0364
  },
  "Temperature": 25.5,
  "humidity": 50,
  "co2": 400,
  "air_quality": {
    "pm25": 10,
    "pm10": 20
  }
}
```

Payload data type: ⓪

JSON

Parameters: ⓪

{ }

[Test Decoder](#)

Output: ⓪

```
{
  "longitude": -77.0364
},
{
  "Temperature": 25.5,
  "humidity": 50,
  "co2": 400,
  "air_quality": {
    "pm25": 10,
    "pm10": 20
  }
}
```

Step 4

Select 'Test Decoder' and the output should match the JSON ingest.

Ways to Extend this Step

Add your own JavaScript to add a random number or more. You can also ask your AI Assistant to write some code for you. Below is a sample to add a random number called 'extra'.

```
function parsePayload(payload, params) {
  payload.extra = Math.random();
  return payload;
}
```

Step 5

Scroll down the page, and you will see the 'Sensor Mapping' element of the page. This is where you map the JSON output to the list of supported sensors for core.aql.com. Core.aql.com will attempt to automap the key to the correct sensor. If any look incorrect, select the correct sensor from the 'Sensor Type' dropdown menu in the 'Sensor Type' column of the 'JSON sensor output mapping' table.

SENSOR KEY	SENSOR TYPE	
co2	CO2	Attach Sensor
humidity	Humidity	Attach Sensor
latitude	Latitude	Attach Sensor
longitude	Longitude	Attach Sensor
pm10	Particulate matter (PM10)	Attach Sensor
pm25	Particulate matter (PM2.5)	Attach Sensor
temperature	Temperature	Attach Sensor
timestamp	Time	Attach Sensor

Once you are happy with the key and the sensor type select 'Attach Sensor' for each sensor listed as an output from the decoder parsing.

SENSOR KEY	SENSOR TYPE	
co2	CO2	Detach
humidity	Humidity	Detach
latitude	Latitude	Detach
longitude	Longitude	Detach
pm10	Particulate matter (PM10)	Detach
pm25	Particulate matter (PM2.5)	Detach
temperature	Temperature	Detach
timestamp	Time	Detach

Once you have attached all the sensors finally hit the SAVE button.

Step 6

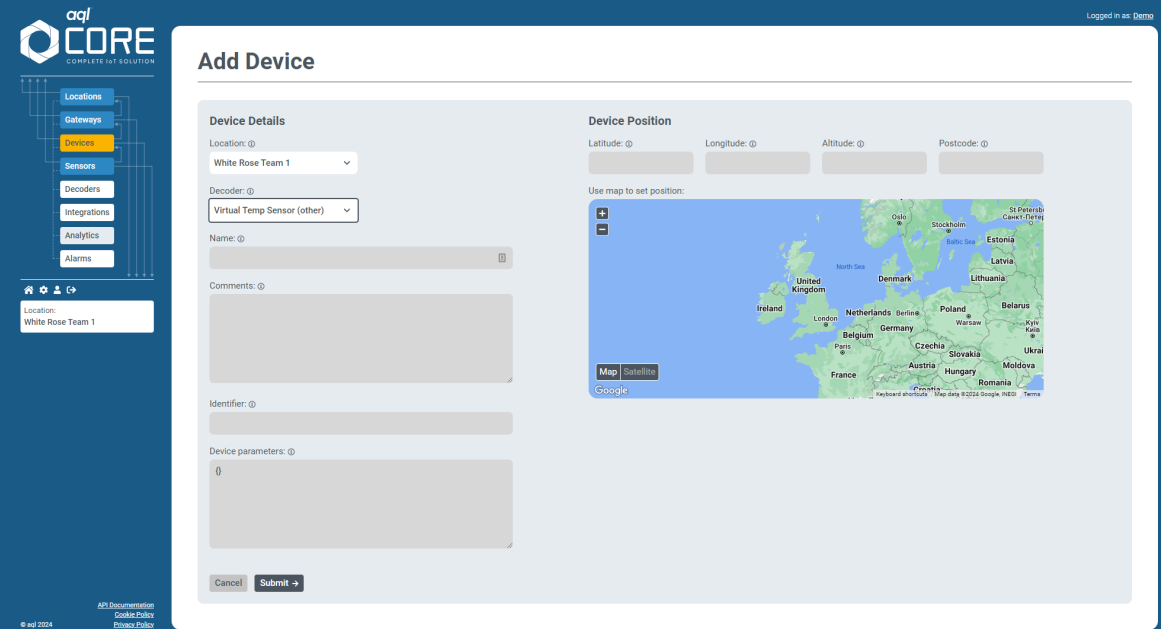
Next, select 'Devices' from the left-hand menu. Select your team from the location dropdown, and select 'Add Device'.



WOODHOUSE
GROVE SCHOOL

aq/ HACKATHON


Step 7



- Select your newly created decoder from the dropdown menu, then enter a name and comment that describes your virtual device.
- Enter a unique identifier, for example Team1-Virtual-Temp-Device.
- Under 'Device Position', enter the school's GPS location or postcode.
- Finally, select Save and your new device is ready to test

You will now see your new virtual device, with all the sensors we created in the earlier decoder. The url of this page should look like 'https://core.aql.com/devices/9BQngfGY896'. That last part is the device ID for your new device. Copy it and save it somewhere safe for a later exercise.

Note: If you add further sensors to your decoder, these will appear automatically when the initial reading is uploaded.



Locations

Gateways

Sensors

Decoders

Integrations

Analytics

Alarms

Location:
White Rose Team 1

Device:
Virtual Temp Sensor

Virtual Temp Sensor

[Edit device](#)
[Delete data](#)
[Generate Certificate](#)

Decoder:
Virtual Temp Sensor

Identifier:
VIRTUAL TEMP SENSOR

Date Added:
16/04/2024

Type:
Other

Decoder Comments:
Virtual Temp Sensor

Last Updated:
N/A

Comments:
Virtual Temp Sensor

Sensors

SENSOR	KEY	ALARMS	LATEST VALUE	LAST UPDATED	DATA RETENTION
CO2	co2	-	N/A	N/A	90 days
Humidity	humidity	-	N/A	N/A	90 days
Latitude	latitude	-	N/A	N/A	90 days
Longitude	longitude	-	N/A	N/A	90 days
Particulate matter (PM10)	pm10	-	N/A	N/A	90 days
Particulate matter (PM2.5)	pm25	-	N/A	N/A	90 days
Temperature	temperature	-	N/A	N/A	90 days
Time	timestamp	-	N/A	N/A	90 days

Event Markers

Add Event Type

Marker type:
None

Add date:

Step 8

We are now going to upload a sample reading for your new device using the Core IoT API. So that the API knows which user you are when you upload a reading the API request requires an API Bearer Token.

We are now going to create a bearer token so we can upload an initial reading. Select the 'settings' icon in the bottom left of the nav bar. This is represented by the Cog Icon.

Step 9

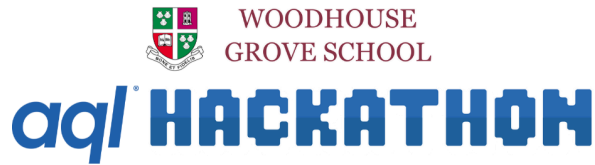
The bearer tokens are located at the bottom of the Settings page. Select the 'Add bearer token', after which you will be asked to choose a name as a reference for your bearer token. For example Team1-Security-Token.

Bearer Tokens

[Add Bearer Token](#)

TOKEN NAME	DATE CREATED	LAST USED
------------	--------------	-----------

Once the token has been created, you'll be redirected to a new page with your bearer token visible at the top. Be sure to copy the token and save it somewhere safe! You will need it later on for this challenge and further challenges that use the Core IoT API.



Step 10

We are now going to upload a reading to your new virtual sensor using a Web Request. This can be easily completed using Curl or platforms such as Postman. The download links for these are

Curl:

<https://curl.se/windows/>

Postman:

<https://www.postman.com/>

We are going to upload a sensor reading for the newly created device. We can use this step for future challenges.

Replace the device ID which is included in the query string and your bearer token from earlier in the snippet below. You can upload up to 100 values with a single request.

The documentation for this request can be found at <https://api.core.aql.com/doc/>

```
curl --location 'https://api.core.aql.com/v1/devices/YOUR_DEVICE_ID/add-reading' \
--header 'Authorization: Bearer REPLACE WITH YOUR BEARER TOKEN' \
--header 'Accept: application/json' \
--header 'Content-Type: application/json' \
--data '{
  "readings": [
    {
      "latitude": 37.7749,
      "longitude": -122.4194,
      "temperature": 2,
      "humidity": 50,
      "co2": 400,
      "pm25": 10,
      "pm10": 20
    },
    {
      "latitude": 37.7749,
      "longitude": -122.4194,
      "temperature": 9,
      "humidity": 50,
      "co2": 400,
```

```

    "pm25": 10,
    "pm10": 20
  }
]
}'

```










A successful submission will return a 201. If the data can be processed, a 422 will be returned.

Step 11

Navigate back to your device http://core.aql.com/devices/YOURDEVICE_ID

You will see the data frames in the logging section and the latest reading value will be displayed.

Sensors

SENSOR	KEY	ALARMS	LATEST VALUE	LAST UPDATED	DATA RETENTION	
CO2	co2	-	400.00 ppm	1 hour ago	90 days	
Humidity	humidity	-	50.00 %	1 hour ago	90 days	
Latitude	latitude	-	37.77	1 hour ago	90 days	
Longitude	longitude	-	-122.42	1 hour ago	90 days	
Particulate matter (PM10)	pm10	-	20.00 µg/m3	1 hour ago	90 days	
Particulate matter (PM2.5)	pm25	-	10.00 µg/m3	1 hour ago	90 days	
Temperature	temperature	-	9.00 °C	1 hour ago	90 days	
Time	timestamp	-	2024.00 s	1 hour ago	90 days	
Value	extra	-	0.20	1 hour ago	90 days	

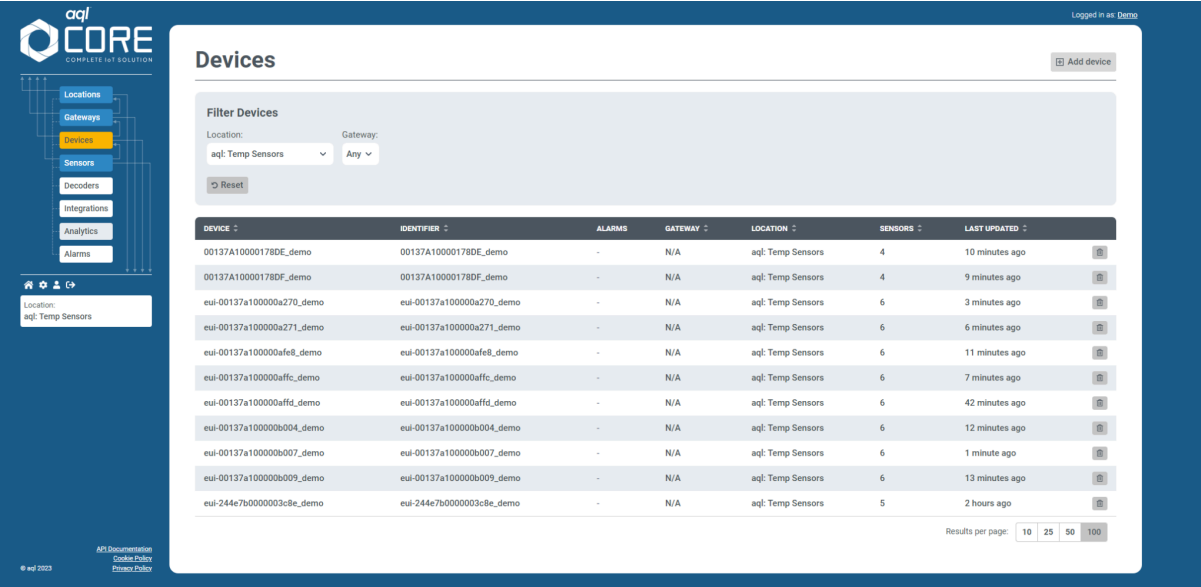
You have now created your virtual sensor and can upload readings. Well done! You can now move on to the next challenge.

Challenge 2: Create a custom IoT Chart

In this challenge, you will choose one of the aql testbed sensors and create a custom chart, and you will make a shareable link that you can share with your friends and family post-event.

Step 1

First, select 'Devices' from the left menu, and browse the list of sensors available. Use filters to narrow down the available devices based on the location assigned to your team.



Devices

Filter Devices

Location: aql: Temp Sensors Gateway: Any

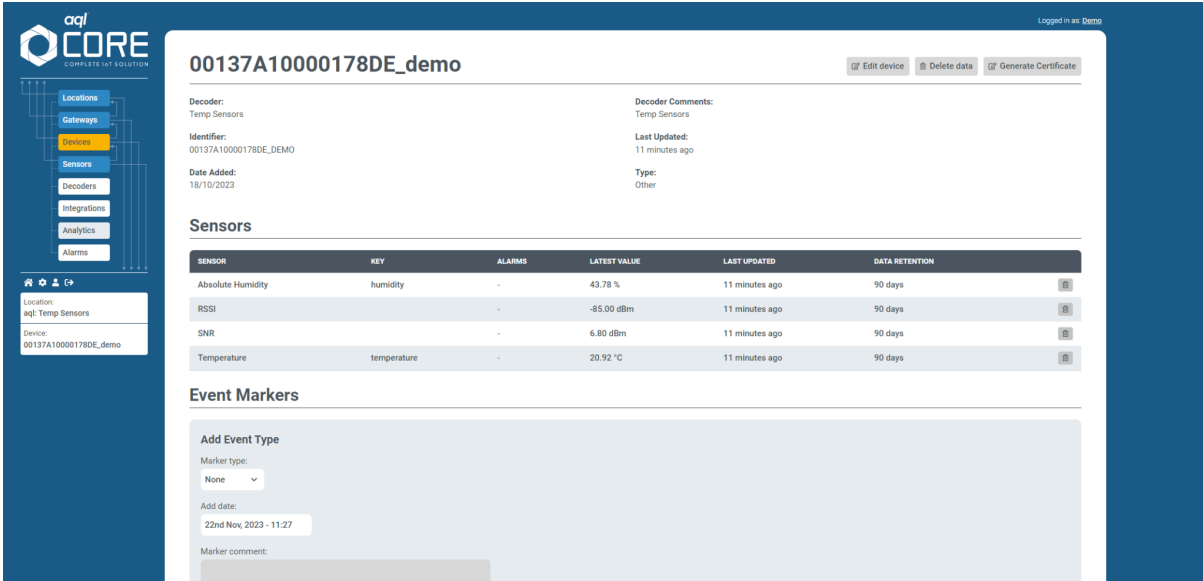
Reset

DEVICE	IDENTIFIER	ALARMS	GATEWAY	LOCATION	SENSORS	LAST UPDATED
00137A10000178DE_demo	00137A10000178DE_demo	-	N/A	aql: Temp Sensors	4	10 minutes ago
00137A10000178DF_demo	00137A10000178DF_demo	-	N/A	aql: Temp Sensors	4	9 minutes ago
eui-00137a100000a270_demo	eui-00137a100000a270_demo	-	N/A	aql: Temp Sensors	6	3 minutes ago
eui-00137a100000a271_demo	eui-00137a100000a271_demo	-	N/A	aql: Temp Sensors	6	6 minutes ago
eui-00137a100000afe8_demo	eui-00137a100000afe8_demo	-	N/A	aql: Temp Sensors	6	11 minutes ago
eui-00137a100000affc_demo	eui-00137a100000affc_demo	-	N/A	aql: Temp Sensors	6	7 minutes ago
eui-00137a100000affd_demo	eui-00137a100000affd_demo	-	N/A	aql: Temp Sensors	6	42 minutes ago
eui-00137a100000b004_demo	eui-00137a100000b004_demo	-	N/A	aql: Temp Sensors	6	12 minutes ago
eui-00137a100000b007_demo	eui-00137a100000b007_demo	-	N/A	aql: Temp Sensors	6	1 minute ago
eui-00137a100000b009_demo	eui-00137a100000b009_demo	-	N/A	aql: Temp Sensors	6	13 minutes ago
eui-244e7b0000003c8e_demo	eui-244e7b0000003c8e_demo	-	N/A	aql: Temp Sensors	5	2 hours ago

Results per page: 10 25 50 100

Step 2

Select the device you created earlier from the list view. Keep a note of the device identifier as we are going to use this again in the next challenge.



00137A10000178DE_demo Edit device Delete data Generate Certificate

Decoder: Temp Sensors
Identifier: 00137A10000178DE_DEMO
Date Added: 18/10/2023
Decoder Comments: Temp Sensors
Last Updated: 11 minutes ago
Type: Other

Sensors

SENSOR	KEY	ALARMS	LATEST VALUE	LAST UPDATED	DATA RETENTION
Absolute Humidity	humidity	-	43.78 %	11 minutes ago	90 days
RSSI	-	-	-85.00 dBm	11 minutes ago	90 days
SNR	-	-	6.80 dBm	11 minutes ago	90 days
Temperature	temperature	-	20.92 °C	11 minutes ago	90 days

Event Markers

Add Event Type

Marker type:
 None
 Add date:
 22nd Nov, 2023 - 11:27
 Marker comment:

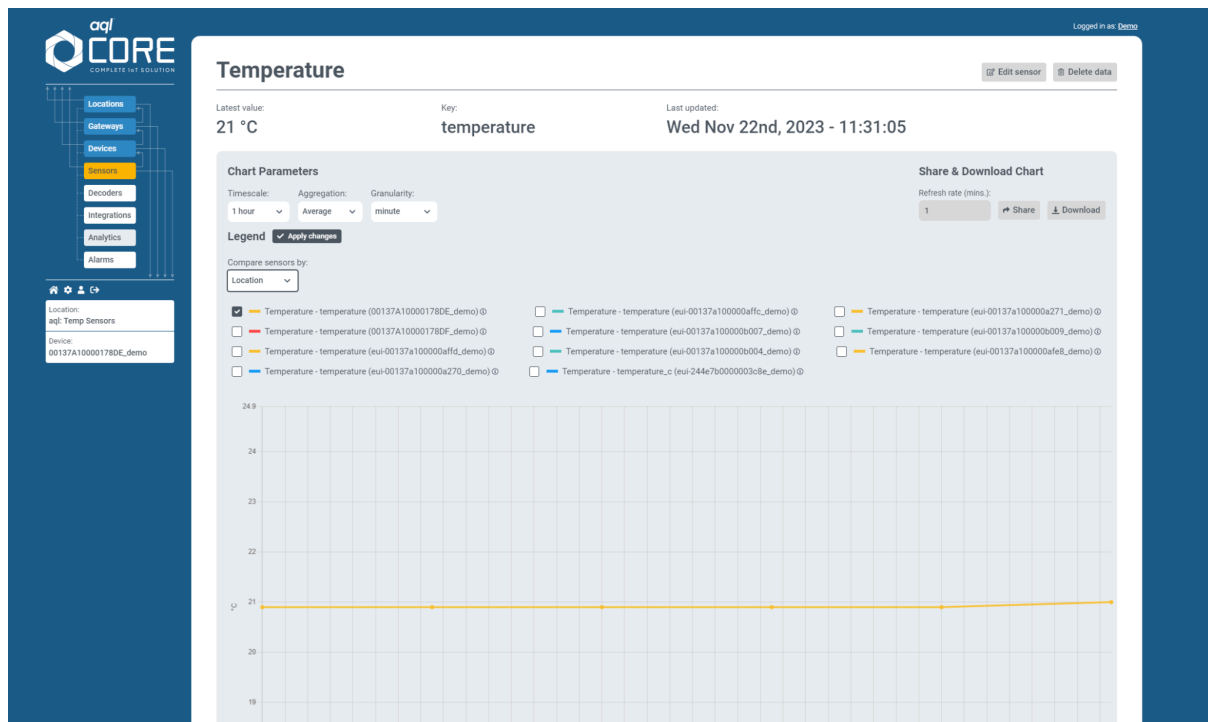
Step 3

You will now see the device summary screen. This will display the list of sensors that the device handles. Next, select the sensor you wish to build a custom chart for.

Step 4

When you select the sensor it will display the latest readings. In the top left corner of the chart you will see the timescale, aggregation and granularity. Adjust these to see the readings over your specific period.

At this stage, if you want, you can add further sensors from a device, or location by expanding the legend.



Step 4

Next, choose the refresh rate for the shared widget and press the share widget before finally opening the chart.

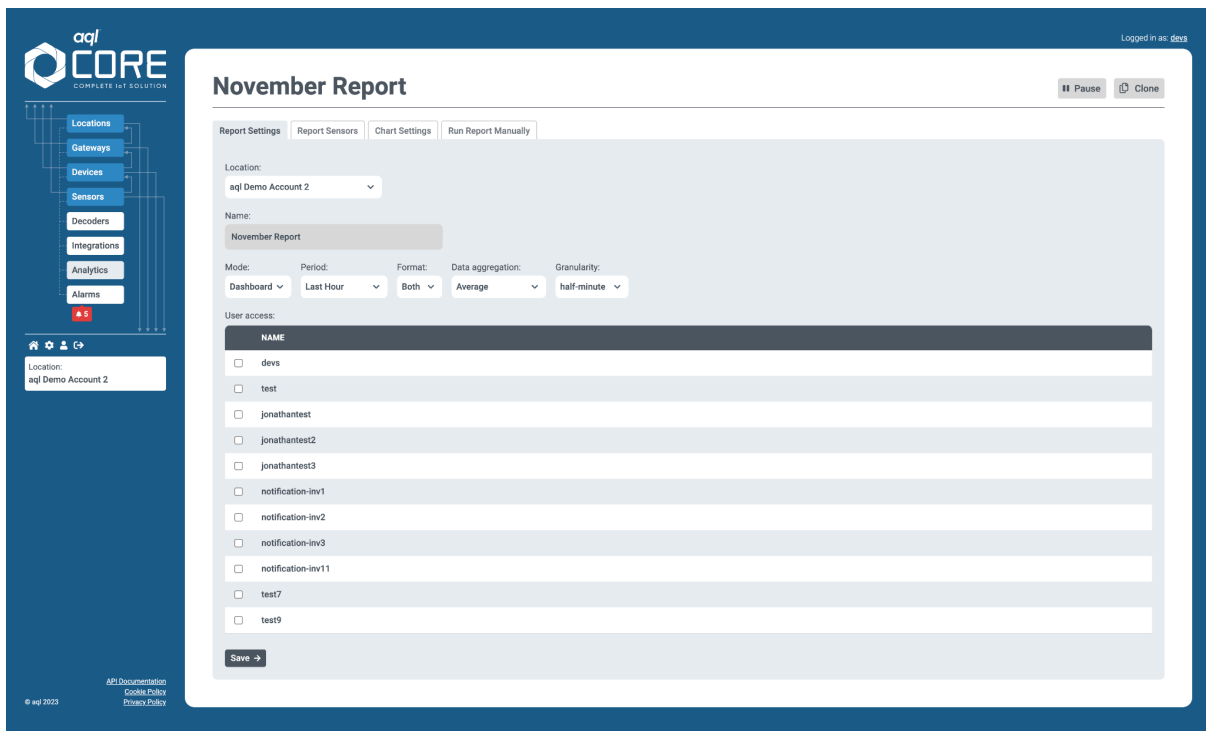
You have now completed this challenge and this chart can be shared post-event.

Challenge 3: Create custom IoT Analytic Widgets

In this challenge, you will use the sensor you selected from Challenge 2 and you will create a Dashboard Widget for your team's location dashboard.

Step 1

- Select the 'Analytics' section from the menu on the left,
- Select the 'Add Analytics Report' button in the top right of the page.
- Enter a name and description for your team's analytics widget and select a desired location from the location field. It's important to select the location that has the devices you wish to use for this challenge.
- Ensure the mode is set to 'Dashboard'
- Choose the data period, data aggregation, and granularity you want to use for your widget
- Click the 'Save' button



The screenshot shows the 'aql CORE' interface. On the left is a navigation menu with options: Locations, Gateways, Devices, Sensors, Decoders, Integrations, Analytics, and Alarms. The 'Analytics' section is highlighted. The main area is titled 'November Report' and includes tabs for 'Report Settings', 'Report Sensors', 'Chart Settings', and 'Run Report Manually'. The 'Report Settings' tab is active, showing a form for configuring the report. The 'Location' is set to 'aql Demo Account 2'. The 'Name' is 'November Report'. The 'Mode' is 'Dashboard', 'Period' is 'Last Hour', 'Format' is 'Both', 'Data aggregation' is 'Average', and 'Granularity' is 'half-minute'. Below these settings is a 'User access' section with a table of users and checkboxes for selection. The users listed are: devs, test, jonathantest, jonathantest2, jonathantest3, notification-inv1, notification-inv2, notification-inv3, notification-inv11, test7, and test9. A 'Save' button is at the bottom of the form.

Step 2

We will now choose the sensors and the colours of the widget elements.

- Select the 'Sensors' tab
- In the 'Actions' section on the right half of the page, select the 'Add Sensors' button.
- The sensor selection window will then appear
- With this sensor selection window visible you can select your device from challenge 2. When you select your device you will notice that the sensors will filter to show only the sensors from this device.



aql HACKATHON

- Select the sensors you wish to display in your widget. You select these by selecting them in the sensors section.
- Next, select the colour you require from the colour picker. Ensure to press the '**Select colour**' from the colour picker.
- Next select 'Add Sensors'. You will see the sensors in the chart sensors list view.

November Report

Report Settings | Report Sensors | Chart Settings | Run Report Manually

Gateways: Select... | Devices: Select... | Sensor Types: Select... | Sensors: Select...

multi purpose 4G IoT platform
multi purpose 4G IoT platform
Tessst 33 Testing long device name text collapse
Testing long device name text collapse
Test
Test 3 test
Test 3 test
Test 3 test

Search

\$GPGBS
\$GPGGA
\$GPRRS
\$GPGSA
\$GPGST
\$GPGSV
\$GPHDT
\$GPLLQ

Absolute Humidity
Accelerometer
Accelerometer X
Accelerometer Y
Active or void
Age of Differential GPS Data Record
Air Temperature in Degrees Celsius
Altitude

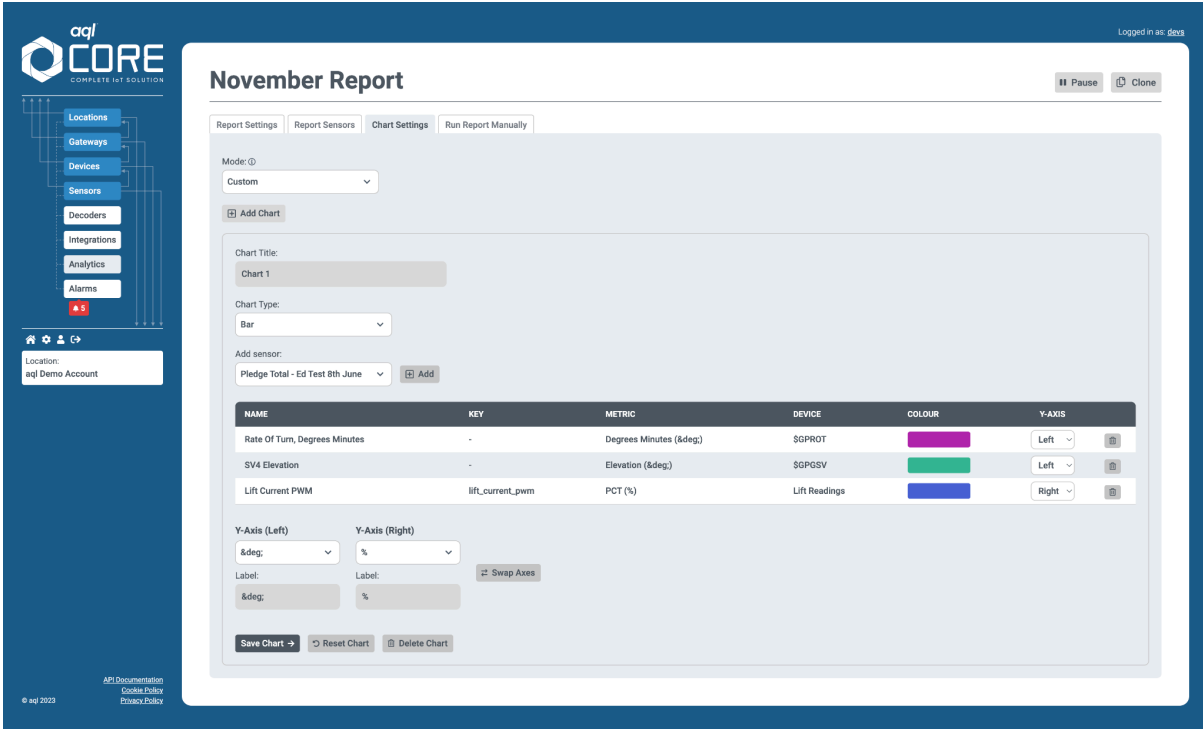
Absolute Humidity (ans)
Absolute Humidity (asdasd)
Absolute Humidity (humidity...)
Absolute Humidity (test)
acc1 (acc3)
Accelerometer (accelerometer.g)
Accelerometer X (accelerometer_x.g)
Accelerometer X (accelerometer_x.g)

Add Sensor ->

SENSOR	DEVICE	KEY	METRIC	VALUE	LAST UPDATED	COLOR
Lift Current PWM	Lift Readings	lift_current_pwm	Percent	-	N/A	
SV4 Elevation	\$GPGSV	-	Elevation	-	N/A	
Rate Of Turn, Degrees Minutes	\$GPROT	-	Degrees Minutes	-	N/A	
Pledge Total	Ed Test 8th June	-	Count Integer	-	N/A	

Step 5

Finally, select the charts tab and select the type of chart you wish to render. If you have chosen multiple sensors you can add two Y axes to the widget. Select 'Custom' under the mode dropdown for increased functionality, name the chart, make necessary amendments and then select "Save" to save the chart.



November Report Pause Clone

Report Settings | Report Sensors | **Chart Settings** | Run Report Manually

Mode: Custom

+ Add Chart

Chart Title: Chart 1

Chart Type: Bar

Add sensor: Pledge Total - Ed Test 8th June + Add

NAME	KEY	METRIC	DEVICE	COLOUR	Y-AXIS
Rate Of Turn, Degrees Minutes	-	Degrees Minutes (°)	\$GPROT	 	Left ⌵
SV4 Elevation	-	Elevation (°)	\$GPGSV	 	Left ⌵
Lift Current PWM	lift_current_pwm	PCT (%)	Lift Readings	 	Right ⌵

Y-Axis (Left): ° ⌵ Y-Axis (Right): % ⌵ ⚡ Swap Axes

Label: ° ⌵ Label: % ⌵

Save Chart ↗ Reset Chart ⌵ Delete Chart

© aql 2023 API Documentation Cookie Policy Privacy Policy

Step 6

- Now, select locations from the location menu within the left-hand menu.
- Select the location you used for your widget.
- You will see your new widgets within your location dashboard.
- You can create a custom dashboard by adjusting the size and position of the widgets.
- If you have chosen multiple devices, when you select the device from the devices widget, it will filter your widget to show the device you have selected.

Well done! Challenge 3 is now complete.

Challenge 4: Create a Live Twitter Integration

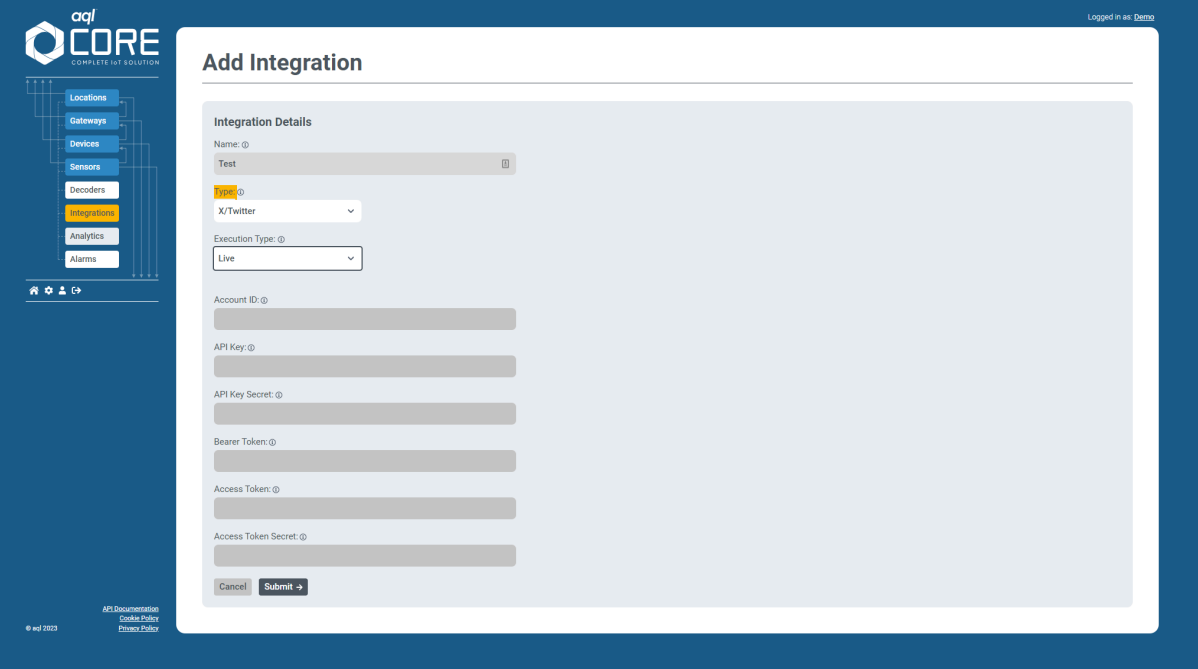
In this challenge, you are going to create an X/Twitter integration that will post a custom Tweet with your chosen sensor readings. The tweets will appear on our aql Dev X/Twitter account (@DevAql37849).

Step 1

Select “integrations” from the left menu

Step 2

- Select ‘Add Integration’, and you will then see the below form to create an integration.
- Before entering the API details, enter a unique name for the integration, select X/Twitter as the integration type, and set the execution type to scheduled. Due to the possibility of lots of sensors sending readings through X/Twitter posts on the event day, we suggest selecting 5 minutes as the schedule time and limiting the amount of sensors you enable with the integration (see step 3):



Next, enter our Dev X/Twitter account API Details

- AccountID: 378491647888965649743872
- API Key: XKHIVXIUfBOve4wnsT6VQKHII
- API Key Secret: t7gKQsQFnpMe43cGGImiW4ZOuaR9q6ZdAfcDQ138ZcKFGTHVE4
- Bearer Token:
AAAAAAAAAAAAAAAAAAAAAKMjmwEAAAAA%2FU%2B7ak%2FRofKR1hUcpgqIB%2FXRHwY%3DpLDyDHGmQ32tAgPfKGz6zoHchFkXrBsZVSeAFBpxJHEtM7vGLR
- Access Token: 1646868244408532992-ggHmtf3tpXHXwABQYBiUXiZ1ccBZg9
- Access Token Secret: qCiYDAbYa7ofivxZSADe8vqgwldZMNhCxsrbxLb2Tlfy

Step 3

- Enable the sensors you wish to use with the integration by toggling the 'Integrate' button. You can use the sensor you selected in challenge 2.
- To enable the sensor select the Device from the filters section.
- Locate the sensor in the list view and select the integrate column. When you select the sensor a Tick will appear.
- You have now chosen the sensor we plan to post to Twitter/X. We now need to create the post template.

Sensors

Filter Integrations
 Location: Any Gateway: Any Device: Any
Reset

INTEGRATE	SENSOR	KEY	DEVICE	LATEST VALUE	LAST UPDATED	DATA RETENTION
<input checked="" type="checkbox"/>	Particulate matter (PM2.5)	mc_pm2_5	White Rose Park Air Quality Sensor	55 µg/m3	17th Nov, 2023 - 08:38	90 days
<input type="checkbox"/>	Absolute Humidity	humidity	00137A10000178DF_demo	45.9 %	17th Nov, 2023 - 08:41	90 days
<input type="checkbox"/>	Absolute Humidity	humidity	00137A10000178DE_demo	36.2 %	17th Nov, 2023 - 08:44	90 days
<input type="checkbox"/>	Absolute Humidity	ambient_humidity	647FDA000000AC7D_demo	52.0 %	17th Nov, 2023 - 08:42	90 days
<input type="checkbox"/>	Absolute Humidity	ambient_humidity	647FDA000000AC9D_demo	57.5 %	17th Nov, 2023 - 08:44	90 days
<input type="checkbox"/>	Absolute Humidity	ambient_humidity	647FDA000000AB4D_demo	97.0 %	17th Nov, 2023 - 08:46	90 days
<input type="checkbox"/>	Accelerometer X	accelerationx	00137A10000033AB_demo	0.000 g	17th Nov, 2023 - 06:55	0 days
<input type="checkbox"/>	Accelerometer Y	accelerationy	00137A10000033AB_demo	0.000 g	17th Nov, 2023 - 06:55	0 days
<input type="checkbox"/>	Accelerometer Z	accelerationz	00137A10000033AB_demo	0.000 g	17th Nov, 2023 - 06:55	0 days
<input type="checkbox"/>	Analog Input Voltage	Aux2 Voltage	1D24E7_demo	3.30 V	17th Nov, 2023 - 08:37	0 days

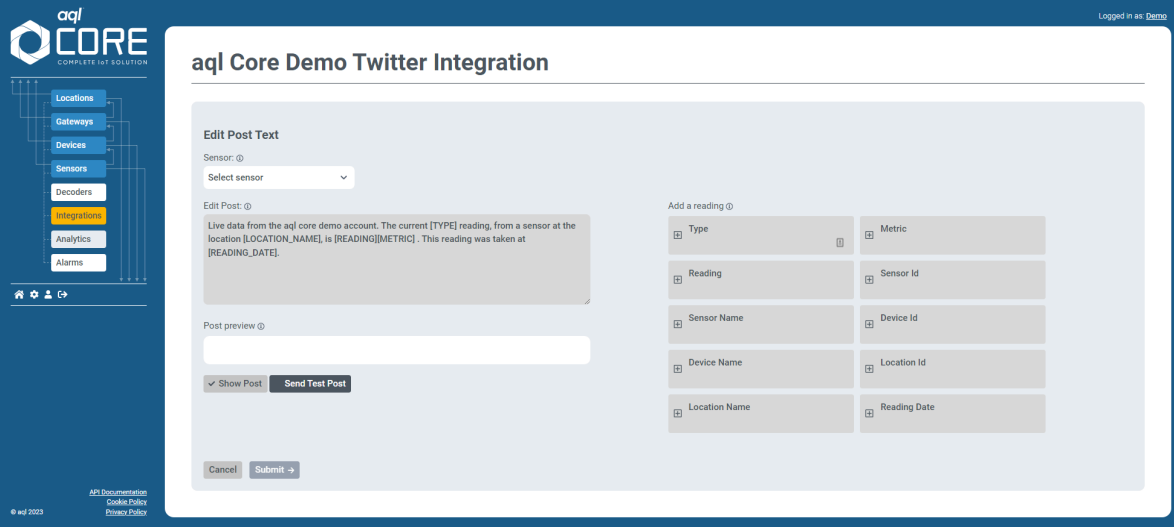
Showing 1 to 10 of 1382 results

<
1
2
3
4
5
6
7
8
9
10
...
138
139
>

Results per page:
 10
25
50
100

Step 4

- Select 'Edit Post' to change the message that appears in the X/Twitter post. You can enter a custom social media post. On the right half of the screen, you can choose to add dynamic values to your post such as sensor name, reading metric etc.
- You can add these values to your posting by clicking the plus values will allow you to insert variable values into the body of the post. These change depending on values with each sensor reading that triggers the integration.
- You can select a sensor to display example values and click 'Show Post' to display how it will appear in the post on X/Twitter.



aql Core Demo Twitter Integration

Edit Post Text

Sensor: @
Select sensor

Edit Post: @
Live data from the aql core demo account. The current [TYPE] reading, from a sensor at the location [LOCATION_NAME], is [READING][METRIC]. This reading was taken at [READING_DATE].

Post preview @

☒ Show Post

Add a reading @

Type	Metric
Reading	Sensor Id
Sensor Name	Device Id
Device Name	Location Id
Location Name	Reading Date

Review your post and click 'Submit' to save your changes.

Step 5


Once saved, you will be returned to the integration summary page. To enable your integration select 'Enable Integration' in the top right corner.





WOODHOUSE
GROVE SCHOOL

aql HACKATHON

Logged in as: Demo

 Enable

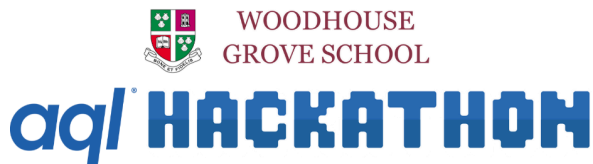
 Edit Integration

 Edit Post

Bearer Token:

Access Token:

Well done! You have now completed this challenge and your posts for the hackathon can be seen on: <https://twitter.com/DevAql37849>



Challenge 5: Use Python to read a sensor's latest reading

Our Core IoT platform API provides the ability to control all aspects of the platform including retrieving data from a single or selection of sensors. With this challenge, you will create a Python script to query the latest value from a selection of sensors and will display these to the console.

If you want to run the Python locally you can jump to Step 4. Otherwise, if you want to build as a Docker image, you can follow the following steps

<https://code.visualstudio.com/docs/languages/python>

<https://code.visualstudio.com/docs/python/python-tutorial>

Step 1

Go to the official Docker website: <https://www.docker.com/products/docker-desktop>
Download Docker Desktop for your operating system (Windows, macOS, or Linux).
Follow the installation instructions provided for your operating system.

Verify Docker installation:

After installing Docker Desktop, open a terminal or command prompt.

Install Docker desktop onto your machine.

Step 2

Create a new folder for your Python challenges.

Step 3

Create a **Dockerfile** and add the following elements.

```
# Use an official Python runtime as a parent image
FROM python:3.10-slim

# Set the working directory in the container
WORKDIR /app

# Copy the requirements file into the container at /app
COPY requirements.txt /app/

# Install any needed dependencies specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Copy all Python files from the host's hackathon folder into the container's
/app folder
COPY hackathon/*.py /app/
```

Step 4

Next, create a **requirements.txt** file and add the following contents.

```
matplotlib
requests
```

Step 5

Create a file called **compose.yml** and add the following contents.

```
services:
  python-app:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "80:80"
    volumes:
      - ./hackathon:/app
```

Step 6

Create a subfolder called **hackathon**. This is where you will save all of your Python files. Create a file in this subfolder called **hello.py** and add the following code.

```
# hello.py

def main():
    print("Hello, world!")

if __name__ == "__main__":
    main()
```

Step 7

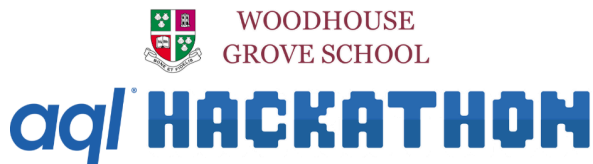
Open a command prompt and run the following command:

```
docker compose run python-app python hello.py
```

This will build the Docker file initially. You should see the following output:

Hello, world!

Step 8



Log into <https://core.aql.com/login> and create a bearer token if you haven't created one from an earlier challenge. Copy this as you will need it for the API request.

Step 9

Choose a series of sensor IDs from the sensors list - we will need these IDs for the coding element. You can access these IDs by clicking on the 'Sensors' option on the left hand side of the page, selecting the location assigned to your team, then copying the unique identifier from the end of the URL:

`https://core.aql.com/sensors/SENSOR_ID`

Step 10

Create a new Python file in your hackathon subfolder called challenge5.py.

Step 11

Enter the code below, then replace the token and IDs with the chosen values from the earlier steps.

```
import requests

# Constants
CORE_URL = 'https://api.core.aql.com/v1/'
TOKEN = 'YOUR_TOKEN_ID'
SENSOR_IDS = [
    "REPLACE_WITH_SENSOR_ID",
    "REPLACE_WITH_SENSOR_ID"
]
```

Step 12

Let's add the loop to handle each sensor.

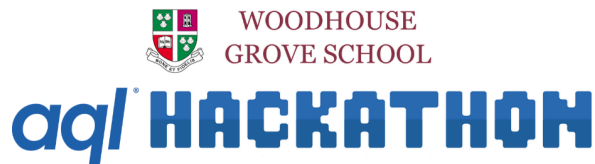
```
# Get data from each station
for sensor_id in SENSOR_IDS:
    print("Getting data from", sensor_id)
```

Run the challenge using:

```
docker compose run python-app python challenge5.py
```

You should see the sensor IDs echoed to the console output.

Step 13



Add a definition block to query

https://api.core.aql.com/doc/#api-Sensor-Sensor_data/latest with the bearer token created earlier.

```
def get_reading(sensor_id):
    url = f"{CORE_URL}sensors/{sensor_id}/sensor-data/latest"
    headers = {'Authorization': f'Bearer {TOKEN}'}
    try:
        response = requests.get(url, headers=headers)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.HTTPError as e:
        print(e)
```

Add the following updates to the initial loop:

```
reading = get_reading(sensor_id)
print(reading)
```

Run the code using:

```
docker compose run python-app python challenge5.py
```

The latest readings should be displayed on the screen.

The final code should look like this, with your bearer token and sensor IDs

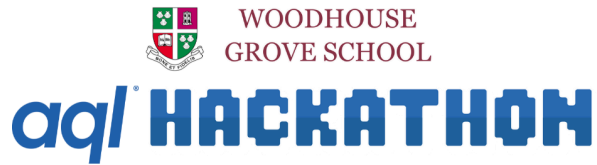
```
import requests

# Constants
CORE_URL = "https://api.core.aql.com/v1/"
TOKEN = "YOUR BEARER TOKEN"
SENSOR_IDS = ["YOUR_SENSOR_ID", "YOUR_SENSOR_ID"]

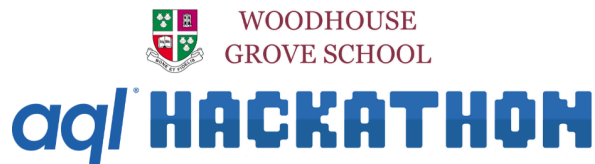
def get_reading(sensor_id):
    url = f"{CORE_URL}sensors/{sensor_id}/sensor-data/latest"
    headers = {"Authorization": f"Bearer {TOKEN}"}
    try:
        response = requests.get(url, headers=headers)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.HTTPError as e:
        print(e)

# Get data from each station
for sensor_id in SENSOR_IDS:
    reading = get_reading(sensor_id)
    print(reading)

print("Challenge Complete")
```



You have now completed this challenge. You could extend this challenge by calling https://api.core.aq.com/doc/#api-Sensor-Sensor_data/aggregate and returning a range of data and then displaying these values to the screen.



Challenge 6: Extend your virtual sensor to generate random sensor readings using Python

This challenge will see you develop a Python script to generate random sensor values for the virtual sensor that you created in challenge 1. The random sensor readings will upload the JSON sensor readings to core.aql.com using the <https://api.core.aql.com/>

Step 1

Install Docker or configure your local machine to run Python as per **Challenge 5** or run the scripts directly from your local machine.

Step 2

Log into the aql Core IoT platform (<https://core.aql.com/login>) with your provided event credentials.

Step 3

Locate the device ID for the virtual sensor created in Challenge 1. The device ID can be captured from the query string by selecting the device and capturing the string from the url. For example <https://core.aql.com/devices/Yx6LZfr65Ov> , in this example Yx6LZfr65Ov is the device ID. You will need to locate your virtual device and copy the device id using the same approach.

Step 4

Log into <https://core.aql.com/login> and create a bearer token if you haven't created one from an earlier challenge copy this as you will need this for one of the following steps. challenge 1 has the details for how to create a bearer token

Step 5

Create a new Python file in the hackathon folder called challenge6.py

Step 6

Add the the imports and constants, replace with your Bearer token and device ID to your challenge6.py file

```
import requests
import random
```

```
import time

# Constants
CORE_URL = 'https://api.core.aql.com/v1/'
TOKEN = 'REPLACE WITH YOUR TOKEN'
DEVICE_ID = 'REPLACE WITH YOUR BEARER TOKEN'
```

Step 7

Add the definition to generate a random sensor reading, and update this to match the sensor types you created in challenge 1, the array keys match so make sure the key in the JSON object matches the keys in the decoder you created in challenge 1. If you created a sensor type which isn't listed below you can ask the AI assistant to write Python code for your chosen sensor type.

```
def generate_reading():
    """
    Generate a random reading.
    """
    timestamp = int(time.time() * 1000) # Current timestamp in milliseconds
    latitude = round(random.uniform(-90, 90), 4) # Random latitude between
-90 and 90
    longitude = round(random.uniform(-180, 180), 4) # Random longitude
between -180 and 180
    temperature = round(random.uniform(-20, 40), 1) # Random temperature
between -20 and 40
    humidity = random.randint(0, 100) # Random humidity between 0 and 100
    co2 = random.randint(300, 1000) # Random CO2 level between 300 and 1000
ppm
    pm25 = random.randint(0, 50) # Random PM2.5 level between 0 and 50 µg/m³
    pm10 = random.randint(0, 100) # Random PM10 level between 0 and 100 µg/m³

    return {"reading": {
        "timestamp": timestamp,
        "latitude": latitude,
        "longitude": longitude,
        "temperature": temperature,
        "humidity": humidity,
        "co2": co2,
        "pm25": pm25,
        "pm10": pm10
    }}
}
```

Step 8

Next, add the definition block to submit the random reading to the `api.core.aql.com` endpoint

```
def submit_reading(payload):  
    """  
    Submit a reading to the API.  
    """  
    url = f"{CORE_URL}devices/{DEVICE_ID}/add-reading"  
    try:  
        headers = {'Authorization': f'Bearer {TOKEN}', 'Content-Type':  
'application/json'}  
        response = requests.post(url, json=payload, headers=headers)  
        response.raise_for_status()  
        print("Reading submitted successfully:", payload)  
    except requests.exceptions.RequestException as e:  
        print("Error submitting reading:", e)
```

Step 9

Finally, add the generate and submission definition.

```
def generate_and_submit_readings(num_readings):  
    """  
    Generate and submit a specified number of readings.  
    """  
    for _ in range(num_readings):  
        reading = generate_reading()  
        submit_reading(reading)  
        time.sleep(1) # Sleep for 1 second between readings  
  
# Test the function  
if __name__ == "__main__":  
    num_readings = 10 # Number of readings to generate and submit  
    generate_and_submit_readings(num_readings)
```

You can complete a test of the code by running

```
docker compose run python-app python challenge6.py
```

Step 10

We can then confirm that 10 readings have been uploaded by logging into the aql Core IoT platform (<https://core.aql.com/login>) with your provided event credentials. You can view the sensor readings following the process you completed in challenge 2, if you aren't sure how to complete this element you can ask your AI assistant.

Step 11

Once you have confirmed your readings have been updated, we are going to update the script so it will generate a new set of random values every 5 minutes.

Update the main block and generate readings to the following code

```
def generate_and_submit_readings():
    """ Generate and submit readings every 5 minutes. """
    while True:
        reading = generate_reading()
        submit_reading(reading)
        time.sleep(300) # Sleep for 300 seconds (5 minutes)

# Test the function
if __name__ == "__main__":
    generate_and_submit_readings()
```

This will generate new readings every 5 minutes and upload these to Core IoT platform.

You can complete a test of the code by running

```
docker compose run python-app python challenge6.py
```

Step 12

The final code should look like this, with your deviceId, bearer token and any sensor types you have added to the generate block. Well done you now have a virtual sensor which should run for the remainder of the hackathon.

Final Code

```
import requests
import random
import time

# Constants
CORE_URL = 'https://api.core.aql.com/v1/'
TOKEN = 'YOUR BEARER TOKEN'
DEVICE_ID = 'YOUR DEVICE ID'

def generate_reading():
    """
    Generate a random reading.
    """
    timestamp = time.strftime("%Y-%m-%dT%H:%M:%SZ", time.gmtime()) # Current
    timestamp in UTC format
```



WOODHOUSE
GROVE SCHOOL

aqi HACKATHON

```
latitude = round(random.uniform(-90, 90), 4) # Random latitude between
-90 and 90
longitude = round(random.uniform(-180, 180), 4) # Random longitude
between -180 and 180
temperature = round(random.uniform(-20, 40), 1) # Random temperature
between -20 and 40
humidity = random.randint(0, 100) # Random humidity between 0 and 100
co2 = random.randint(300, 1000) # Random CO2 level between 300 and 1000
ppm
pm25 = random.randint(0, 50) # Random PM2.5 level between 0 and 50 µg/m³
pm10 = random.randint(0, 100) # Random PM10 level between 0 and 100 µg/m³

return {"reading": {
    "timestamp": timestamp,
    "latitude": latitude,
    "longitude": longitude,
    "temperature": temperature,
    "humidity": humidity,
    "co2": co2,
    "pm25": pm25,
    "pm10": pm10
}}

def submit_reading(payload):
    url = f"{CORE_URL}devices/{DEVICE_ID}/add-reading"
    headers = {'Authorization': f'Bearer {TOKEN}', 'Content-Type':
'application/json'}
    # response = requests.post(url, json=payload, headers=headers)
    # response.raise_for_status()
    try:
        response = requests.post(url, json=payload, headers=headers)
        response.raise_for_status()
        print("Reading submitted successfully:", payload)
    except requests.exceptions.RequestException as e:
        print("Error submitting reading:", e)

def generate_reading():
    """ Generate a random reading. """
    timestamp = time.strftime("%Y-%m-%dT%H:%M:%SZ", time.gmtime()) # Current
timestamp in UTC format
    latitude = round(random.uniform(-90, 90), 4) # Random latitude between
-90 and 90
    longitude = round(random.uniform(-180, 180), 4) # Random longitude
between -180 and 180
    temperature = round(random.uniform(-20, 40), 1) # Random temperature
between -20 and 40
    humidity = random.randint(0, 100) # Random humidity between 0 and 100
    co2 = random.randint(300, 1000) # Random CO2 level between 300 and 1000
ppm
    pm25 = random.randint(0, 50) # Random PM2.5 level between 0 and 50 µg/m³
```



WOODHOUSE
GROVE SCHOOL

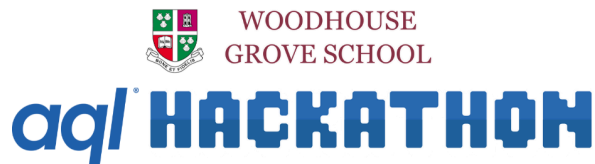
aql HACKATHON

```
pm10 = random.randint(0, 100) # Random PM10 level between 0 and 100 µg/m³
return {
    "reading": {
        "timestamp": timestamp,
        "latitude": latitude,
        "longitude": longitude,
        "temperature": temperature,
        "humidity": humidity,
        "co2": co2,
        "pm25": pm25,
        "pm10": pm10
    }
}

def submit_reading(payload):
    url = f"{CORE_URL}devices/{DEVICE_ID}/add-reading"
    headers = {'Authorization': f'Bearer {TOKEN}', 'Content-Type':
'application/json'}
    try:
        response = requests.post(url, json=payload, headers=headers)
        response.raise_for_status()
        print("Reading submitted successfully:", payload)
    except requests.exceptions.RequestException as e:
        print("Error submitting reading:", e)

def generate_and_submit_readings():
    """ Generate and submit readings every 5 minutes. """
    while True:
        reading = generate_reading()
        submit_reading(reading)
        time.sleep(300) # Sleep for 300 seconds (5 minutes)

# Test the function
if __name__ == "__main__":
    generate_and_submit_readings()
```



Challenge 7: Create a custom web page to display live sensor data

This challenge will see you creating a new webpage, making a javascript call to retrieve a sensor reading, and using the design assets provided to add a header and make a dynamic thermometer reading. You will then use a third-party chart library to plot readings for the last 24 hours. You have complete freedom with this challenge and can follow the steps or create your own virtual sensor design and upload data using the API endpoint in challenge 6

The assets for this challenge can be downloaded from: https://github.com/aql-com/iot_event

Step 1

Log into the aql Core IoT platform (<https://core.aql.com/login>) with your provided event credentials.

You can use the sensor you created in Challenge . Note down the SensorID as we will need this later for the API.

If you created a bearer token in Challenge 1, please jump to Step 4.

Step 2

Select the settings option in the bottom left of the nav bar.

Step 3

Create a new bearer token and allocate a name for this token.

Bearer Tokens			Add Bearer Token
TOKEN NAME	DATE CREATED	LAST USED	

Copy the token as you will need this for the JavaScript later.

Step 4

Choose a location on your computer for the project files for example c:\code

Create a new folder for your project.

Inside the folder, create an HTML file (e.g., `index.html`) and a JavaScript file (e.g., `script.js`).

Step 5

Open the folder in your chosen editor. Visual studio code is free to download from <https://code.visualstudio.com/download>

Open `index.html` in a text editor.

Add HTML structure for your webpage, including elements for displaying the thermometer and temperature details.

Link your JavaScript file (`script.js`) to the HTML file using `<script>` tag. Add any logos and your own colours to the html. A sample can be seen below. Be as creative as you would like at this stage. You can ask the hackathon AI to help you with all these steps.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Thermometer Web Page</title>
  <link rel="stylesheet" href="styles.css"> <!-- You can link a CSS
file for styling -->
</head>
<body>

  <div class="container">
    <h1>Temperature Monitoring</h1>
    <div class="thermometer-wrapper">
      <div id="thermometer-level">
        <!-- Thermometer body -->
        <div id="thermometer-background"></div>
        <!-- Thermometer level -->
        <div id="level" height="0"></div>
      </div>
    </div>
    <div id="temperature"></div> <!-- Temperature details will be
displayed here -->
  </div>
```

```
<script src="script.js"></script> <!-- Link your JavaScript file -->
</body>
</html>
```

Step 6

The HTML above requires CSS to position the live reading over the SVG. You can experiment and add further CSS to add your style to the page.

Add the CSS

```
.thermometer-wrapper {
  height: 400px;
  width: 200px;
  position: relative;
}

#thermometer {
  position: absolute;
}

#thermometer-level {
  position: absolute;
  height: 270px;
  width: 20px;
  left: 85px;
  top: 24px;
}

#thermometer-background {
  position: absolute;
  top: 0;
  left: 0;
  height: 100%;
  width: 20px;
  background: grey;
}
```

```
#level {  
  position: absolute;  
  bottom: 0px;  
  left: 0;  
  width: 20px;  
  background: red;  
}
```

Step 7

Next, we will add the JavaScript to return the latest sensor reading, you will need to replace the Sensor ID and the bearer token. This script makes use of api.core.com. The method:

```
https://api.core.aql.com/v1/sensors/${sensorid}/sensor-data/latest
```

returns the latest reading for a single sensor. There are methods to send an array of sensors which will allow you to load multiple sensors to the UI. Review the documentation and extend the Javascript to use this method or use the sample below.

```
// When the DOM content is loaded  
document.addEventListener("DOMContentLoaded", function() {  
  // API endpoint URL  
  const sensorid = "REPLACE WITH YOUR SENSOR ID";  
  const apiUrl =  
  `https://api.core.aql.com/v1/sensors/${sensorid}/sensor-data/latest`;  
  
  // Bearer token  
  const bearerToken = "REPLACE WITH YOUR BEARER TOKEN";  
  
  // Fetch temperature data from the API  
  fetch(apiUrl, {  
    method: "GET",  
    headers: {  
      "Authorization": `Bearer ${bearerToken}`  
    }  
  })  
  .then(response => response.json())  
  .then(data => {  
    // Get the temperature value from the API response  
  
    const temperature = data.value;
```

```
// Map temperature to thermometer height
const levelHeight = mapTemperatureToHeight(temperature);

// Update thermometer level based on temperature
document.getElementById("level").style.height =
`${levelHeight}px`;

))
.catch(error => console.error("Error fetching temperature:",
error));

});

// Function to map temperature to thermometer height
function mapTemperatureToHeight(temperature) {
  const minTemperature = 0;
  const maxTemperature = 100;
  const maxHeight = 270;

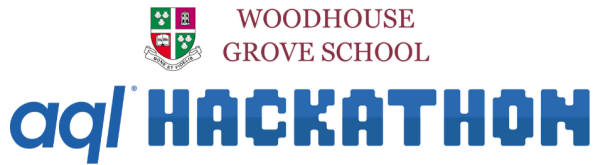
  const normalizedTemperature = Math.max(Math.min(temperature,
maxTemperature), minTemperature);
  const heightPercentage = normalizedTemperature / maxTemperature;
  const levelHeight = maxHeight * heightPercentage;

  return levelHeight;
}
```

Save the file and open the index.html in a browser window. Your value should appear in the visual thermometer. If you right select on the browser and select inspect any JavaScript, errors returned will be displayed there. If you have any problems, you can provide your code and error to your AI Assistant and it will be able to help.

As your virtual sensor uploads temperature readings, you should see your thermometer change. At this stage, you will need to refresh the page to see the latest readings appear.

Step 8



Ask your AI assistant how to extend the sample to add a chart to the page which plots the values for your sensor for today aggregated every hour.

The final code for each class should look like this. It will have your sensor ID and bearer token

Index.html Final Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Thermometer Web Page</title>
  <link rel="stylesheet" href="styles.css"> <!-- You can link a CSS file for
styling -->
</head>
<body>

  <div class="container">
    <h1>Temperature Monitoring</h1>
    <div class="thermometer-wrapper">
      <div id="thermometer-level">
        <!-- Thermometer body -->
        <div id="thermometer-background"></div>
        <!-- Thermometer level -->
        <div id="level" height="0"></div>
      </div>
    </div>
    <div id="temperature"></div> <!-- Temperature details will be displayed
here -->
  </div>

  <canvas id="chart-wrap"></canvas> <!-- Chart will be displayed here -->

  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <script src="script.js"></script> <!-- Link your JavaScript file -->
</body>
</html>
```

Script.js Final Code

```
// When the DOM content is loaded
document.addEventListener("DOMContentLoaded", function() {
  // API endpoint URL
  const sensorid = "Your Sensor ID";
  const apiUrl =
`https://api.core.aql.com/v1/sensors/${sensorid}/sensor-data/latest`;

  // Bearer token
  const bearerToken = "Your Bearer Token";
```



WOODHOUSE
GROVE SCHOOL

aql HACKATHON

```
// Fetch temperature data from the API
fetch(apiUrl, {
  method: "GET",
  headers: {
    "Authorization": `Bearer ${bearerToken}`
  }
})
.then(response => response.json())
.then(data => {
  // Get temperature value from the API response

  const temperature = data.value;

  // Map temperature to thermometer height
  const levelHeight = mapTemperatureToHeight(temperature);

  // Update thermometer level based on temperature
  document.getElementById("level").style.height = `${levelHeight}px`;

})
.catch(error => console.error("Error fetching temperature:", error));

// Fetch chart data
getChartData(bearerToken, sensorid);
});

// Function to map temperature to thermometer height
function mapTemperatureToHeight(temperature) {
  const minTemperature = 0;
  const maxTemperature = 100;
  const maxHeight = 270;

  const normalizedTemperature = Math.max(Math.min(temperature,
maxTemperature), minTemperature);
  const heightPercentage = normalizedTemperature / maxTemperature;
  const levelHeight = maxHeight * heightPercentage;

  return levelHeight;
}

// Function to fetch chart data
const getChartData = (bearerToken, sensorid) => {
  const apiUrl =
`https://api.core.aql.com/v1/sensors/${sensorid}/sensor-data/aggregate/ave`;
  const today = new Date().toISOString().slice(0, 10);
  const body = JSON.stringify({
    "startDate": `${today} 00:00:00`,
    "endDate": `${today} 23:59:59`,
    "granularity": "quarter_hour"
  });
};
```



aql HACKATHON

```
// Fetch chart data from the API
fetch(apiUrl, {
  method: "POST",
  headers: {
    "Accept": "application/json",
    "Authorization": `Bearer ${bearerToken}`,
    "Content-Type": "application/json"
  },
  body: body
})
.then(response => {
  if (response.ok) {
    return response.json();
  } else {
    throw new Error("Failed to fetch chart data");
  }
})
.then(data => {
  processGraphData(data);
})
.catch(error => console.error("Error fetching chart data:", error));
}

// Function to process graph data
const processGraphData = apidata => {
  console.log(apidata);

  if (Object.keys(apidata).length > 0) {
    let graphLabels = [];
    let graphValues = [];

    // Extract graph labels and values from the API response
    for (let i = 0; i < apidata.length; i++) {
      graphLabels.push(apidata[i].sensorReadingDate.substring(11, 16));
      graphValues.push(apidata[i].Average);
    }

    // Create a line chart using Chart.js
    new Chart(
      document.getElementById('chart-wrap'),
      {
        type: 'line',
        data: {
          labels: graphLabels,
          datasets: [
            {
              label: 'Temperature over time',
              data: graphValues,
              borderColor: '#fab400',
            }
          ]
        }
      }
    );
  }
}
```



aql HACKATHON

```
        backgroundColor: '#fab400',  
    }  
    ]  
    }  
    }  
    );  
    }  
}
```

Styles.css Final Code

```
.thermometer-wrapper {  
    height: 400px;  
    width: 200px;  
    position: relative;  
}  
  
#thermometer {  
    position: absolute;  
}  
  
#thermometer-level {  
    position: absolute;  
    height: 270px;  
    width: 20px;  
    left: 85px;  
    top: 24px;  
}  
  
#thermometer-background {  
    position: absolute;  
    top: 0;  
    left: 0;  
    height: 100%;  
    width: 20px;  
    background: grey;  
}  
  
#level {  
    position: absolute;  
    bottom: 0px;  
    left: 0;  
    width: 20px;  
    background: red;  
}
```

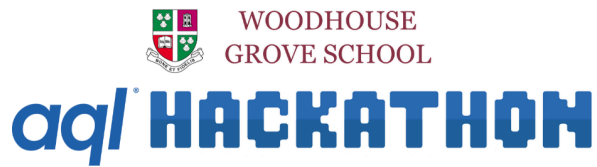
Ways to extend this step

Extend the chart for several days, include other sensor readings, adjust the colour, Use other node charts to plot a scatter or bar chart. With HTML, Javascript and the <https://api.core.aql.com/doc/> you can create rich visualisations. Get creative, we're all excited to see what you can do!



WOODHOUSE
GROVE SCHOOL

aq/ HACKATHON



Challenge 8: Create widgets for your sensors

In challenge 3 you created your first widget using an aql sensor from our 5G IoT testbeds. With this challenge we will create more widgets for each of the sensors you created within your virtual sensor, can you create a bar, line and scatter chart and create your own custom hackathon dashboard to demonstrate all the sensors you have used across the hackathon.