Core team 2022 – documentation

Core teams main tasks were to integrate API communication into data exchange using the cloud service and integration of 3 sensors: Door sensor, Liquid level sensor and water flow sensor.

API:

API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other. We integrated API on Python, Matlab and JavaScript.

Python:

We started with integration on Python because to integrate API on Python is the easiest task out of these three programming languages. The goal of this task was for us to learn how API works.

Matlab:

After that we continued with Matlab and JavaScript. It was needed to integrate API on Matlab so the Smart team could fulfil their goals.

%API access:

client\_id = "Ij7686MXMIXCS4JyGJTe5nyQNQM6w7R9";

client\_secret = "BWbnmLopN6lUDPncNZ5zDmAYvHk81TQUbHWFACt9FwysSfqiBgQYhpkVeIYLtYDQ";

url = "https://api2.arduino.cc/iot/v1/clients/token";

headerFields = {'Content-type' 'application/x-www-form-urlencoded';'charset' 'UTF-8'};

options = weboptions('HeaderFields', headerFields);

response = webwrite(url,...

'client\_id', client\_id,...

'client\_secret', client\_secret,...

'grant\_type','client\_credentials',...

'audience','https://api2.arduino.cc/iot',...

options);

access\_token = response.access\_token;

% Save access token for future calls

headerFields = {'Authorization', ['Bearer ', access\_token]};

options = weboptions('HeaderFields', headerFields, 'ContentType','json','MediaType', 'auto');

%Access lasts for 300 seconds, after 300 seconds token expires, and it is needed to run this code again

*%% 1.*

%Observation of the data with function “timeseries”, it is possible to define the needed time period

%Example:

options.RequestMethod = 'auto'

id(id devicu, device id) - id for sensors: 95e254c8-7421-4d0e-bcb6-4b6991c87b4f

- id for actuators: aa0190a8-9312-4a17-8842-25a1dd483860

pid(id sensor or actuoator ):

-sensors:

- pid svetla(light): d6653286-1d51-49d0-83b9-0dd6bf6b54fe

- pid doors: 67e83f0d-524f-433b-b478-51a5275040af

- pid hum\_bmme: 3b3eccfa-bce9-40d5-a187-a82b75bf300d

- pid hum\_dht: d69a1c3a-0e1f-4461-bc76-727c68e2b4d3

- pid liqud\_state: e979475d-0b5e-474d-9c4f-29a1b5d66c6d

-pid pump: c30dc559-96dc-41d9-9d14-b00cfff2d239

- pid temp\_avg: 8b7294c8-93e9-4ab0-911f-89ffcf3830c4

- pid temp\_bme: a4e7f374-07f2-4071-ba07-f3c96a09a250

- pid temp\_bottom: 68f17374-437c-466d-a7fc-b9b5906002b1

- pid temp\_dht: 567281db-effe-4b49-a907-1e581c463cc0

- pid temp\_top: 61e6a028-5d0d-4ae5-9684-069fed62d784

- pid total\_mL: 974afe32-94cc-4e49-9b74-d7f2eafeae45

- pid water\_detection: 2930af3d-2b63-40c0-b557-f7453254a815

- pid flow\_rate: 179e1ae1-9410-4b6c-9d72-8bf009deb425

- pid hum\_avg: 248bd031-8b74-4923-9264-d36b5659d061

- actuators:

- pid ventilátora(fans): 0c8a7441-7894-4d36-b4ec-7df63dfebd2a

- pid pump: b26776b4-a096-4883-b752-ddf6c89cc3cd

-pid heating: a9f0e988-ad3d-4d77-b0aa-a82abfc70c58

- pid lighting: 128d9d60-0c2f-4a58-a3e4-fb833e10bf81

from=2022-03-30T11:00:00Z

to=2022-03-30T12:00:00Z

interval=1(integer)

tserie = webread("https://api2.arduino.cc/iot/v2/things/{id}/properties/{pid}/timeseries?from=&interval=&to=",options)

tserie = webread("https://api2.arduino.cc/iot/v2/things/{95e254c8-7421-4d0e-bcb6-4b6991c87b4f}/properties/{d6653286-1d51-49d0-83b9-0dd6bf6b54fe}/timeseries?from=2022-03-29T11:00:00Z&interval=1&to=2022-03-29T11:29:00Z",options)

*%% 2.*

%Sending actuators data to server:

options.RequestMethod = 'put'

%Example:

propertyValue = struct('value',130) – it is possible to change the value of actuators

webwrite("https://api2.arduino.cc/iot/v2/things/{id}/properties/{pid}/publish",propertyValue,options)

webwrite("https://api2.arduino.cc/iot/v2/things/{aa0190a8-9312-4a17-8842-25a1dd483860}/properties/{0c8a7441-7894-4d36-b4ec-7df63dfebd2a}/publish",propertyValue,options)

JavaScript:

API integration on JavaScript was on the other hand needed for fulfilling the AppTeam goals.

JavaScript:

API access:

var IotApi = require('@arduino/arduino-iot-client');

var rp = require('request-promise');

async function getToken() {

var options = {

method: 'POST',

url: 'https://api2.arduino.cc/iot/v1/clients/token',

headers: { 'content-type': 'application/x-www-form-urlencoded' },

json: true,

form: {

grant\_type: 'client\_credentials',

client\_id: 'Ij7686MXMIXCS4JyGJTe5nyQNQM6w7R9',

client\_secret: 'BWbnmLopN6lUDPncNZ5zDmAYvHk81TQUbHWFACt9FwysSfqiBgQYhpkVeIYLtYDQ',

audience: 'https://api2.arduino.cc/iot'

}

};

try {

const response = await rp(options);

return response['access\_token'];

}

catch (error) {

console.error("Failed getting an access token: " + error)

}

}

*/\**Access lasts for 300 seconds, after 300 seconds token expires, and it is needed to run this code again\*/

*// 1.*

/\*Observation of the data with function “timeseries”, it is possible to define the needed time period\*/

//Example:

id(id devicu, device id):

- id pre senzory: 95e254c8-7421-4d0e-bcb6-4b6991c87b4f

- id pre akčné členy: aa0190a8-9312-4a17-8842-25a1dd483860

pid(id senzora alebo akčného člena:

-sensors:

- pid svetla(light): d6653286-1d51-49d0-83b9-0dd6bf6b54fe

- pid doors: 67e83f0d-524f-433b-b478-51a5275040af

-pid hum\_bmme: 3b3eccfa-bce9-40d5-a187-a82b75bf300d

- pid hum\_dht: d69a1c3a-0e1f-4461-bc76-727c68e2b4d3

- liqud\_state: e979475d-0b5e-474d-9c4f-29a1b5d66c6d

- pid pump: c30dc559-96dc-41d9-9d14-b00cfff2d239

- pid temp\_avg: 8b7294c8-93e9-4ab0-911f-89ffcf3830c4

- pid temp\_bme: a4e7f374-07f2-4071-ba07-f3c96a09a250

- pid temp\_bottom: 68f17374-437c-466d-a7fc-b9b5906002b1

- pid temp\_dht: 567281db-effe-4b49-a907-1e581c463cc0

- pid temp\_top: 61e6a028-5d0d-4ae5-9684-069fed62d784

- pid total\_mL: 974afe32-94cc-4e49-9b74-d7f2eafeae45

- pid water\_detection: 2930af3d-2b63-40c0-b557-f7453254a815

- pid flow\_rate: 179e1ae1-9410-4b6c-9d72-8bf009deb425

- pid hum\_avg: 248bd031-8b74-4923-9264-d36b5659d061

- (actuators):

- pid ventilátora(fans): 0c8a7441-7894-4d36-b4ec-7df63dfebd2a

- pid pump: b26776b4-a096-4883-b752-ddf6c89cc3cd

- pid heating: a9f0e988-ad3d-4d77-b0aa-a82abfc70c58

- pid lighting: 128d9d60-0c2f-4a58-a3e4-fb833e10bf81

'from': '2022-03-29T11:00:00Z'

'to': '2022-03-29T11:29:00Z'

'interval': 1

//code:

async function run() {

var client = IotApi.ApiClient.instance;

// Configure OAuth2 access token for authorization: oauth2

var oauth2 = client.authentications['oauth2'];

oauth2.accessToken = await getToken();

var prop = new IotApi.PropertiesV2Api(client)

var id = '95e254c8-7421-4d0e-bcb6-4b6991c87b4f'; // {String} The id of the thing

var pid = 'd6653286-1d51-49d0-83b9-0dd6bf6b54fe'; // {String} ID of a numerical property

var opts = {

'from': '2022-03-29T11:00:00Z', // {String} Get data with a timestamp >= to this date (default: 2 weeks ago, min: 1842-01-01T00:00:00Z, max: 2242-01-01T00:00:00Z)

'interval': 1, // {Integer} Binning interval in seconds (defaut: the smallest possible value compatibly with the limit of 1000 data points in the response)

'to': '2022-03-29T11:29:00Z' // {String} Get data with a timestamp < to this date (default: now, min: 1842-01-01T00:00:00Z, max: 2242-01-01T00:00:00Z)

};

prop.propertiesV2Timeseries(id,pid,opts).then(data => {

console.log(data);

},error => {

console.log(error)

});

}

run();

// 2.

//Sending data to server, setting up the actuators:

//Exaple:

propertyValue = {'value':130} - the value of an actuator can be changed here

async function send\_value() {

var client = IotApi.ApiClient.instance;

// Configure OAuth2 access token for authorization: oauth2

var oauth2 = client.authentications['oauth2'];

oauth2.accessToken = await getToken();

var val = new IotApi.PropertiesV2Api(client)

var id = 'aa0190a8-9312-4a17-8842-25a1dd483860'; // {String} The id of the thing

var pid = '0c8a7441-7894-4d36-b4ec-7df63dfebd2a';

var propertyValue = {'value':40}; // 0-255

val.propertiesV2Publish(id, pid, propertyValue).then(function() {

console.log('API called successfully.');

}, function(error) {

console.error(error);

});

}

send\_value()

ESP32 DevKitC:

ESP32-DevKitC is a low-footprint and entry-level development board that is part of the ESP32 series. This board has a rich peripheral set. This is the microcontroller attached to our VESNA eco greenhouse.

A picture containing electronics, circuit

Description automatically generated

Codes that we want to run on ESP32 microcontroller are added to the sketch in IoT cloud. Sketch is accessible as follows: Login to **IoT cloud → Things → VESNA\_sense → Sketch**. We can also add variables in **Things → Setup** .

Arduino Uno:

Arduino Uno is another microcontroller board. It has 14 digital input/output pins, 6 analog inputs, a USB connection, a power jack, an ICSP header and a reset button.

A close-up of a circuit board

Description automatically generated with medium confidence

The use we found for Arduino Uno board during completing our tasks is to relieve ESP32 from another two sensors. Flowrate and Liquid state sensors are connected to Arduino and the code for variables liquidState, flowRate and water\_detection runs here. ESP32 on VESNA gets these values via RX/TX communication between these two microcontrollers.

RX/TX communication:

This is a serial communication between two microcontrollers (ESP32 and Arduino Uno in our case). We connect them with 3 male-to-male jumper wires. One wire is used to connect GND pins on both boards. Second wire connects RX pin of one board with TX pin of another. Third wire connects TX pin of one board with RX pin of another.

A screenshot of a computer

Description automatically generated with low confidence

We used virtual RX and TX pins on 10 and 11 pins on each board using SoftwareSerial softSerial(10,11) command. Next step is to determine which microcontroller is going to be master (ESP32) and which one is going to be slave (Arduino Uno). Slave board has the code to get flowrate, liquidState and water\_detection variables and sends them on request from its master. Master board sends request to slave board and receives wanted information on variables and sends them to cloud.

Arduino IDE (Version 1.8.19):

The Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus.

Graphical user interface, text, application

Description automatically generated

It connects to the Arduino hardware to upload programs and communicate with them. Arduino IDE can be used for uploading codes on both Arduino Uno and ESP32 microcontrollers. It uses C++ programming language.

Door Sensor (MC-38):

The MC-38 magnetic switch sensor comes in a pair consisting of a Reed Switch and a magnet. The wired part of the module is the reed switch, whereas the other part is the magnet.

A picture containing text

Description automatically generated

The MC-38 can be connected to a microcontroller. The Reed Switch part has 2 outputs- Ground and Signal. We connected this sensor to pin 19 on ESP32. Using digitalRead function we get True of False (1 or 0) values for variable doors. True represents door on VESNA being opened, False represents them being closed.

Liquid level sensor (XKC-Y25-NPN):

XKC-Y25-NPN is a non-contact sensor for liquid level detection. It can detect liquid levels of various toxic substances, acid, alkali, and all kinds of liquid in high pressure airtight container. It can be widely used and has no special requirement for liquid medium and material of container.

A picture containing cable, gauge, adapter

Description automatically generated

We connected it to digital pin 5 on Arduino Uno board. Using digitalRead function we get variable liquidState that can acquire values True or False (1 or 0). True represents that water is present, and we can see a red light on liquid level sensor. False represents lack of water and light is off. Arduino waits for request from its master and then sends liquidState value to ESP32.

Water flow sensor (YF-S201):

This sensor contains a pinwheel sensor to measure how much liquid has moved through it. There's an integrated magnetic hall effect sensor that outputs an electrical pulse with every revolution. The hall effect sensor is sealed from the water pipe and allows the sensor to stay safe and dry. The sensor comes with three wires: power that is connected to 5V pin, ground connected to GND pin and Hall effect pulse output connected to pin 2 on Arduino Uno board.

A picture containing device, gauge

Description automatically generated

By counting the pulses from the output of the sensor, you can easily calculate water flow. We calculated water flow in litres per minute and save it to variable flowRate. Pump sends water for through water flow sensor and we count the sum of all values of flowRate. If this sum is bigger than 30 000 we can say that there was enough water that went through the water flow sensor and there is no problem. Otherwise there was not enough water and that tells us something is wrong. We save this True or False (1 or 0) value as variable water\_detection, final variable from our work with sensors.