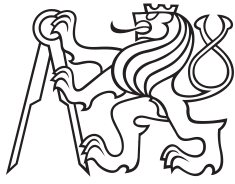


**Bachelor Project**



**Czech  
Technical  
University  
in Prague**

**F3**

**Faculty of Electrical Engineering  
Department of Telecommunication engineering**

## **Tuya cloud-based IoT device control system**

**Martin Jordán**

**Supervisor: doc. Ing. Leoš Boháč, Ph.D.  
May 2023**



## I. Personal and study details

Student's name: **Jordán Martin** Personal ID number: **499248**  
Faculty / Institute: **Faculty of Electrical Engineering**  
Department / Institute: **Department of Telecommunications Engineering**  
Study program: **Electronics and Communications**

## II. Bachelor's thesis details

Bachelor's thesis title in English:

**Tuya Cloud-based IoT Device Control System**

Bachelor's thesis title in Czech:

**System ovládání IoT za řízení přes cloud Tuya**

Guidelines:

The aim of the bachelor's thesis is to practically verify the way of connecting originally manufactured devices (sensors and actuators) to the Tuya IoT cloud and also to enable their monitoring and control via a classic WEB browser and mobile application. In the practical part, the student is required to design and implement a gateway that allows the connection of APC 230 V power outlets that are controlled via a terminal interface.

The outcome of the work will be:

- 1) a detailed description of the implementation of a generic sensor and actuator connected to the Tuya cloud with detailed instructions, both on the device side and on the Tuya cloud side. Primarily use Python as the programming tool.
- 2) IoT Tuya gateway for remote control of APC 230V power outlets. Select appropriate hardware for the gateway and write appropriate software
- 3) Web interfaces to allow switching of the above outlets
- 4) Mobile apps enable switching of the above outlets.

Bibliography / sources:

[1] A. K. Gupta and R. Johari, 'IOT based Electrical Device Surveillance and Control System,' 2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU), 2019, pp. 1-5, doi: 10.1109/IoT-SIU.2019.8777342.

[2] A. R. Biswas and R. Giaffreda, 'IoT and cloud convergence: Opportunities and challenges,' 2014 IEEE World Forum on Internet of Things (WF-IoT), 2014, pp. 375-376, doi: 10.1109/WF-IoT.2014.6803194.

[3] SMART, Gary. Practical Python Programming for IoT: Build advanced IoT projects using a Raspberry Pi 4, MQTT, RESTful APIs, WebSockets, and Python 3. Birmingham: Packt Publishing, 2020. ISBN 1838982469.

Name and workplace of bachelor's thesis supervisor:

**doc. Ing. Leoš Bohá , Ph.D. Department of Telecommunications Engineering FEE**

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **17.01.2023** Deadline for bachelor thesis submission: \_\_\_\_\_

Assignment valid until: **22.09.2024**

\_\_\_\_\_  
doc. Ing. Leoš Bohá , Ph.D.  
Supervisor's signature

\_\_\_\_\_  
Head of department's signature

\_\_\_\_\_  
prof. Mgr. Petr Páta, Ph.D.  
Dean's signature

### III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

\_\_\_\_\_  
Date of assignment receipt

\_\_\_\_\_  
Student's signature

## Acknowledgements

In this part, I would like to thank my supervisor doc. Ing. Leoš Boháč, Ph.D. for his time and consultation.

## Declaration

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, May 26, 2023

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 26. května 2023

## Abstract

The aim of the thesis is to propose and implement method for connecting of originally manufactured devices to Tuya IoT cloud platform. These devices are mostly controlled via non-standard interfaces and don't use control logic usual to home automation systems and smart homes. The thesis will describe in detail the procedure for setting up a generic device on the Tuya cloud side as well as the implementation of communication interface on the device side. The interface for communication between the Tuya cloud and the general device will be implemented into Python module.

In the practical part, an IoT gateway will be implemented, which will enable control and monitoring of 230V APC sockets with a telnet communication interface through the Tuya cloud platform. A control interface will be implemented into a web application to control the connected device.

**Keywords:** MQTT, Internet of Things, Tuya, cloud, IoT

**Supervisor:** doc. Ing. Leoš Boháč,  
Ph.D.

## Abstrakt

Cílem této práce je navrhnout a realizovat připojení nestandardních zařízení k IoT cloudové platformě Tuya. Tato zařízení jsou většinou ovládána prostřednictvím nestandardních rozhraní a obvykle nepoužívají řídicí logiku běžnou pro systémy domácí automatizace a inteligentní domácnosti. V práci bude detailně popsán postup pro nastavení obecného zařízení na straně cloudu Tuya i implementace komunikačního rozhraní na straně zařízení. Rozhraní pro komunikaci mezi cloudem Tuya a obecným zařízením bude implementována do Python modulu.

V praktické části bude implementována IoT brána, která umožní ovládání a monitorování 230V APC zásuvek s telnetovým komunikačním rozhraním skrze cloudovou platformu Tuya. K ovládání zařízení bude implementováno ovládací rozhraní do webové aplikace.

**Klíčová slova:** MQTT, Internet věcí, Tuya, cloud, IoT

**Překlad názvu:** Systém ovládání IoT zařízení přes cloud Tuya

# Contents

<b>1 Introduction</b>	<b>1</b>	8.3.1 Overview	38
<b>2 Assignment analysis</b>	<b>3</b>	8.3.2 Authorization	38
2.1 Connection method	3	8.3.3 Service API	39
2.2 Hardware selection	3	8.3.4 Assets	39
2.3 Web and mobile application	4	8.3.5 Devices	40
<b>3 MQTT protocol</b>	<b>5</b>	8.4 API explorer	42
3.1 MQTT protocol introduction	5	<b>9 API request structure and authentication</b>	<b>45</b>
3.2 MQTT control packets	5	9.1 Introduction	45
<b>4 Device model</b>	<b>9</b>	9.2 Request structure	45
4.1 Properties	9	9.3 Authentication method	45
4.2 Events	9	<b>10 Web control interface</b>	<b>47</b>
4.3 Actions	9	10.1 Introduction	47
<b>5 Tuya MQTT client implementation</b>	<b>11</b>	10.2 Project structure	47
5.1 Introduction	11	10.3 Authentication mechanism	47
5.2 DeviceID and DeviceSecretID	11	10.4 Tuya API calls implementation	49
5.3 Client implementation	11	10.4.1 Cross-Origin Resource Sharing (CORS)	49
5.4 Using the Tuya MQTT client module	15	10.4.2 Tuya API calls implementation	49
<b>6 Setting up a new device on the Tuya cloud platform</b>	<b>17</b>	10.5 User interface	53
6.1 Setting up a new device on Tuya cloud	17	<b>11 Linking device to Tuya app</b>	<b>55</b>
6.1.1 Introduction	17	11.1 Introduction	55
6.1.2 Creating a product	17	11.2 Linking devices to Tuya Smart Life application	55
6.2 Product development menu	20	<b>12 Conclusion</b>	<b>57</b>
6.2.1 Function definition	21	<b>Bibliography</b>	<b>59</b>
6.2.2 Device development menu	24		
6.2.3 Device management	25		
6.2.4 Online debugging	26		
6.2.5 Application development	28		
6.2.6 Product configuration	29		
6.2.7 Finishing product configuration	31		
<b>7 Developing a gateway for APC Telnet power strip integration.</b>	<b>33</b>		
7.1 Introduction	33		
7.2 Telnet interface of the power strip	33		
7.3 Implementation	34		
<b>8 Cloud development</b>	<b>37</b>		
8.1 Introduction	37		
8.2 Cloud project setup	37		
8.3 Sections of the Cloud development menu	38		

## Figures

2.1 Diagram of communication between devices. ....	4
3.1 QoS levels in MQTT communication .....	7
6.1 Product development menu on Tuya IoT platform .....	18
6.2 Device type selection screen ....	18
6.3 Development method selection screen .....	19
6.4 Product information configuration screen .....	20
6.5 Function definition .....	21
6.6 Standard functions pop-up menu	22
6.7 Creating a custom function ....	23
6.8 Adding parameter to the Event or Action .....	24
6.9 Choosing development method .	24
6.10 Device management menu ....	25
6.11 Registering a single device ....	26
6.12 Online debugging menu .....	27
6.13 Debugging Actions .....	28
6.14 Panel development tab of the Application development menu ...	29
6.15 Scenario connection settings ...	30
6.16 Data parsing menu .....	31
8.1 Cloud development menu .....	38
8.2 Overview tab of the cloud development menu .....	38
8.3 Asset setup menu .....	40
8.4 Linking Tuya Smart Life app account to the cloud project .....	41
8.5 Linking devices from Tuya Smart Life account .....	42
8.6 API explorer .....	43
10.1 Authentication process with JWT tokens .....	48
10.2 User interface of the web application .....	53
11.1 Control interface of the APC power strip in Tuya Smart Life app	56
11.2 Linking a device to the Tuya Smart Life application .....	56

## Tables

3.1 CONNACK return codes .....	6
--------------------------------	---





# Chapter 1

## Introduction

*Tuya IoT Platform* is a Platform as a Service IoT (Internet of Things) cloud solution. Platform as a service model is a cloud computing model, where cloud service provider provides hardware (cloud server) and software layer on which can customer build their cloud application [2]. Tuya IoT platform provides complete cloud, networking and smartphone app support for smart devices, such as smart home appliances. Tuya can be used to control and monitor smart devices through smartphone applications. There are many devices being sold for the Tuya system (e.g. smart light bulbs, smart power outlets, smart light switches), that can be used to set up a smart home. In this thesis, I will discuss how to connect devices with non standard interfaces to the Tuya IoT cloud platform. These devices originally do not have an interface to communicate with Tuya.

In this thesis, I will be mainly focusing on connecting devices to the Tuya cloud platform via MQTT protocol. MQTT protocol is one of the most popular protocol for IoT applications, due to its ease of implementation and ability to run on resource constrained devices and limited network bandwidth. Tuya MQTT client will be implemented into Python module, which can be imported into any Python code and which will provide easy implementation of Tuya cloud connectivity on any device, that can run Python code and has TCP/IP stack. A complete methodology for configuring a smart device on the Tuya cloud will be described in this thesis.

To control devices, Tuya Smart Life mobile application from the Tuya ecosystem will be used. Additionally, a control interface will be implemented into a web application to control the connected device. The interface will use API requests to communicate with Tuya cloud platform.



## Chapter 2

### Assignment analysis

In this chapter, I will analyse the assignment for this thesis and I will discuss possible solutions to the tasks set in the assignment.

#### 2.1 Connection method

There are two methods to connect devices to Tuya cloud: TuyaOS and TuyaLink. TuyaOS is an operating system based on RTOS and Linux. The downside of using TuyaOS is the necessity to use Tuya's proprietary hardware (microcontrollers and network modules). To allow wider selection of hardware to be used, I will be using TuyaLink connection method, which uses MQTT protocol with defined format of JSON message payload to communicate between Tuya and the device.

#### 2.2 Hardware selection

As for the hardware, I am using Raspberry Pi model 3B equipped with Raspbian operating system. Raspberry Pi will be used to create the IoT gateway for remote control of APC 230V power outlets. There are many choices for a single-board computer to choose from. I chose Raspberry Pi, because I had it readily available, but any other single-board computer, such as many of Raspberry Pi clones (e.g. Orange Pi, Banana Pi) would work just fine, as long as it does have Wi-Fi or Ethernet connectivity and enough computing power to run Python. Another advantage of Raspberry Pi is the abundance of interfaces, such as Bluetooth, Wi-Fi, USB, SPI, I2C and digital I/O pins. Due to this, it is possible to integrate many types of devices to the Tuya system, such as USB thermometers, many types of sensors, that use SPI or I2C bus (e.g. temperature, humidity sensors) or BLE (Bluetooth Low Energy) sensors. It is also possible to use CC2531 Zigbee coordinator and use the Raspberry Pi as a gateway for Zigbee devices.

Interesting opportunity for future work would be to rewrite the implemented Tuya client Python module to MicroPython language, which would allow it to run on smaller microcontrollers such as Raspberry Pi Pico or Pyboard for less resource heavy applications. MicroPython is an efficient

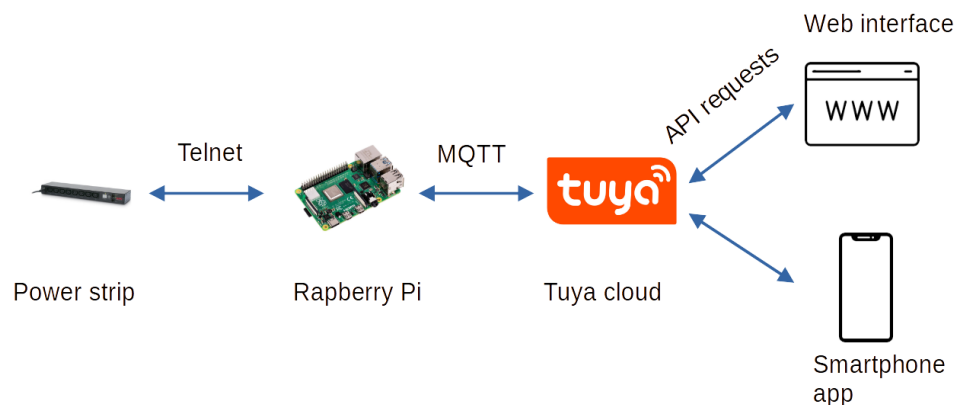
implementation of Python designed to run on resource constrained micro-controllers. Python is generally not designed to run on smaller devices than single-board computers, as it is quite inefficient in its implementation. To port the module to MicroPython, I would have to rewrite the Tuya client module with different MQTT library, such as umqtt library.

## 2.3 Web and mobile application

The web interface will be based on the Express framework. The web interface will use API calls to the API endpoints of the Tuya cloud to switch on/off the power outlets

As for the mobile application the Tuya Smart life application will be used for switching of the power outlets. Another option would be to develop my mobile application from ground up and implement API calls for communication between the cloud and the mobile application, but it is much easier to use application, which is already integrated into the ecosystem of Tuya and just load my configuration for the APC power outlets into it.

The goal of this thesis will be to implement the communication between the devices and interfaces, as shown on [Fig. 2.1].



**Figure 2.1:** Diagram of communication between devices.

## Chapter 3

### MQTT protocol

#### 3.1 MQTT protocol introduction

*MQTT* (Message Queue Telemetry Transport) is a lightweight communication protocol run over TCP/IP. MQTT is based on *publish-subscribe model*, with the MQTT *broker* as a central point for handling the message exchange. MQTT messages are sorted into topics. Client devices can either publish messages to the topic, which means they send a message to the broker, or they can subscribe to the topic, which means the broker sends them every message it receives in the same topic. MQTT protocol has become a popular choice for IoT applications or Machine to Machine (M2M) communication due to its ease of implementation and ability to run on resource constrained devices. MQTT protocol is payload agnostic, and has small transport overhead, which makes it suitable for applications, where network bandwidth is a valuable resource [11]. Client usually connects to the broker via TCP on port 1883, via WebSockets on ports 8080/8081, or via TLS on port 8883.

MQTT topics are hierarchical, hierarchy levels are separated in the name of the topic with a slash symbol "/" [10], topic name can be for example: *home/living\_room/lights/light\_1*. If the client publishes or subscribes to a topic, that isn't already registered at the broker, the broker must create a new topic with that name.

#### 3.2 MQTT control packets

MQTT communication consists of exchange of MQTT *control packets*. Each control packet has a fixed header and some types of control packets also have a variable header and a payload (packets used for acknowledgment of message delivery usually don't have a payload). First four bits of the fixed header are MQTT control packet type identifier, which identifies the type of control packet being sent [12].

Client requests connection to the broker with a CONNECT control packet. CONNECT packet header contains name of the protocol, connect flags specifying the communication behaviour and for indicating presence of optional fields in payload, *keep alive* time and protocol level [13]. Protocol level speci-

ifies the version of MQTT protocol being used, this parameter must match the version, that broker is using in order to connect, most commonly used version is MQTT v3.1.1. Keep alive specifies the time period after which is the client disconnected from the broker, if the broker doesn't receive any message in this period. If the broker doesn't receive any packet during 1.5 times the keep alive time period, it must disconnect the client. Payload of the CONNECT packet must contain ClientId (client identifier) as the first field. ClientId is an UTF-8 encoded string, that must be unique to each client connected to the broker and identifies client to the broker. Other optional parts of the payload, which must be indicated in the header of the CONNECT packet by connect flags, are username, password, WILL topic and WILL message. Username and password are used to further authenticate the client device to the broker, these parameters must match the username and password, that is set for the client at the broker, otherwise the client will be disconnected. WILL message is a message that gets stored at the broker and gets published into the WILL topic upon disconnection of the client that created it.

Successful connection is confirmed from broker by CONACK (connection acknowledge) control packet. In the CONACK packet's variable header is stored return code [14], which is used to indicate, whether the connection was successful [Table 3.1].

Value	Return Code Response	Description
0	0x00 Connection Accepted	Connection accepted
1	0x01 Connection Refused, unacceptable protocol version	The Server does not support the level of the MQTT protocol requested by the Client
2	0x02 Connection Refused, identifier rejected	The Client identifier is correct UTF-8 but not allowed by the Server
3	0x03 Connection Refused, Server unavailable	The Network Connection has been made but the MQTT service is unavailable
4	0x04 Connection Refused, bad user name or password	The data in the user name or password is malformed
5	0x05 Connection Refused, not authorized	The Client is not authorized to connect

**Table 3.1:** CONNACK return codes

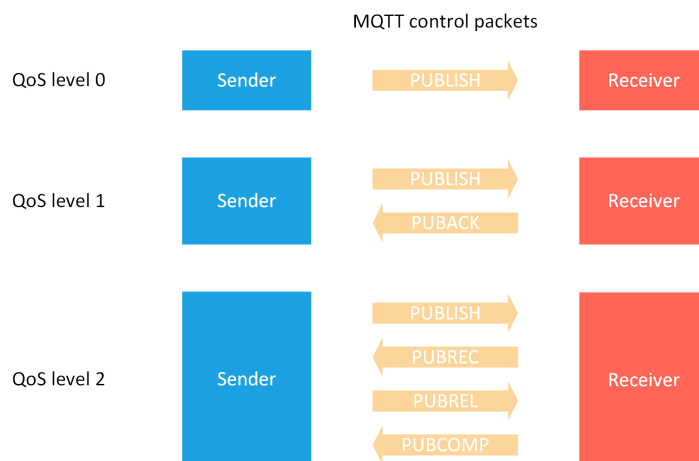
If the client doesn't have any control packet to send during the keep alive period, it can send PINGREQ (ping request) packet. PINGREQ can be also used to verify whether the connection between the client and the broker is working. Broker responds to PINGREQ packet with a PINGRESP (ping response) packet.

To publish a message, client must send PUBLISH control packet. PUBLISH packet's header contains QoS level, retain flag and DUP flag [15].

Retain flag = 1 indicates to the broker, that it should store the PUBLISH message it receives and resend it to the other clients, that will subscribe to the same topic in the future, retain flag = 0 means, that broker should discard the message after sending it to all currently subscribed clients. DUP flag is used to indicate, whether this is the first delivery of the packet (DUP = 0), or this is redelivery of an earlier attempt to send the packet. PUBLISH packet's payload contains the published message in any format. PUBLISH packet is acknowledged by the broker according to the selected QoS level, as shown in [Figure 3.1].

To subscribe to the topic, client sends SUBSCRIBE control packet. The payload of the SUBSCRIBE packet contains the list of the topics, to which client wants to subscribe. Subscription to the topic is confirmed by the broker by SUBACK (subscribe acknowledgement) control packet.

QoS (Quality of Service) in MQTT defines what measures are taken on the client side and the broker side to guarantee the delivery of the message. In MQTT protocol there are defined 3 QoS levels [16]. QoS level 0 doesn't guarantee delivery of the message, sender sends PUBLISH packet exactly once, receiver doesn't send PUBACK acknowledgement packet, which means PUBLISH packet is delivered either exactly once or not at all. QoS level 1 guarantees, that message is delivered at least once. With QoS level 1 receiver sends PUBACK (publish acknowledgement) packet to sender after receiving PUBLISH packet. QoS level 2 guarantees, that message is delivered exactly once, it is the strictest QoS level. It has the largest transport overhead of all three QoS levels. After receiving the PUBLISH control packet message, receiving side will send PUBREC (publish receive) packet, sending side will discard PUBLISH packet and send PUBREL (publish release) packet, on which the receiving side responds with PUBCOMP (publish complete) packet to end the communication.



**Figure 3.1:** QoS levels in MQTT communication

The session is ended with a DISCONNECT control packet sent from the

### 3. MQTT protocol

---

client to the broker.



## Chapter 4

### Device model

Device model is an abstraction of physical device. For purposes of interfacing and connecting a device to the Tuya cloud, device model is defined as a set of *properties*, *actions* and *events*.

#### 4.1 Properties

Device properties describe the state of a device and can be of various data types, such as numerical value, float, string, bool, date, array and others. Property can be used to update the cloud about the state of a device, or to control a function of the device from the cloud, for example a smart light bulb can have properties like brightness, colour and on/off state. Each device property can be configured as send-only (property values can be only sent from the cloud to the device), report-only (property values can be only sent from the device to the cloud), or send and report (property values can be sent both ways).

#### 4.2 Events

Events are report-only notifications to the cloud from the device. They are especially useful for reporting alerts and faults in the device, for example they could be triggered by overheat or overcurrent protection in the device. Every event can be accompanied by any number of output parameters. Output parameters can be of same data types as properties (bool, value, float, string and others) and can be used to further specify the parameters of the reported event.

#### 4.3 Actions

Actions are instruction messages from the cloud to a device to perform more complex task. Actions work on request-response model of communication, target devices should return the result of the action performed to the cloud in action response message. Actions are not intended for changing device properties. When using actions, it is important to configure devices to

properly send action response message to each to action message the device receives. Outcoming action messages from the cloud can be accompanied by any number of input parameters, which can be used to further specify the action to be performed. Action response messages can be accompanied by any number of output parameters. Output and input parameters of actions can be defined with same data types as properties, or output parameters of events (bool, value, float, string and others).

## Chapter 5

# Tuya MQTT client implementation

### 5.1 Introduction

In this chapter, I will describe the process of implementation of *Tuya MQTT client* module, which will be used to handle the communication between Tuya cloud and the gateway device (Raspberry Pi). This software will be implemented into Python module, so the Tuya cloud connectivity can be added to any device, that can run Python, by simply importing the module

Tuya cloud server, which acts as a MQTT broker in the context of MQTT communication, uses MQTT protocol to communicate with client devices. Eclipse *Paho MQTT* Python library will be used to manage MQTT communication and Tuya client will be built on its basis. Paho MQTT library is one of the most popular Python libraries for MQTT protocol. Paho provides all functions, that are needed to handle MQTT communication, such as connecting to the broker, publishing and subscribing to the topics, or callbacks for handling incoming messages.

Tuya uses JSON as a format of the payload of the MQTT messages. JSON (JavaScript Object Notation) [8] is a lightweight data-interchange format made of key:value pairs, that is both easy to read for humans and parse and generate for computers.

### 5.2 DeviceID an DeviceSecretID

After registering the device at Tuya IoT platform, two unique identifiers are assigned to each device from the Tuya cloud: DeviceID (device identifier) and DeviceSecret (device secret identifier). These are used for the authentication process at the start of the MQTT communication. How to configure and register a device on the Tuya cloud and obtain DeviceID and DeviceSecretID identifiers is described in detail in chapter 6.

### 5.3 Client implementation

Tuya MQTT client will be implemented in *TuyaClient* class within the Python module. At first, I created a constructor to initialize TuyaClient class with

DeviceID, DeviceSecretID, which will be obtained upon registration of the device at the Tuya developer platform, and path to TLS certificate, which can be downloaded from Tuya IoT Developer Platform [20].

```

1 class TuyaClient:
2     def __init__(self, device_ID, device_secret_ID, cert_path):
3         self.device_ID = device_ID
4         self.device_secret_ID = device_secret_ID
5         self.cert_path = cert_path

```

**Listing 5.1:** Class constructor

The `client_init(self, func=None, *args)` method of the `TuyaClient` class is used to connect and authenticate the client to the MQTT server and connect callback functions for handling incoming messages. The `client_init` function takes 2 input arguments (beside `self`): `func` and `*args`. These allow to pass a function, which will be triggered upon receiving a device property update, this function can be placed outside of the Tuya client module. `*args` parameter represents any number of parameters of said function. This will be used later when implementing gateway on the Raspberry Pi for APC power strip to send a command to the interface of the Telnet interface of the power strip after receiving a device property update from the cloud.

MQTT client instance is created with `mqtt.Client(ClientID)` function of the Paho library on line 2.

As a next step, `username` and `password` are set. These will be used upon connection to authenticate the device to the cloud. Username is simply concatenated string [5] from `deviceID`, signature method 10-digit current, timestamp with following syntax: `"deviceID"|signMethod=hmacSha256,timestamp="10-digit current timestamp",secureMode=1,accessType=1`.

Password is set as a signature created using the HMAC-SHA256 cryptographic method, where hashed content is `content_string`, which is defined by Tuya with following format: `deviceId="deviceID",timestamp="10-digit current timestamp",secureMode=1,accessType=1`, and the key is the `DeviceSecretID` obtained after registration of a device on Tuya developer platform [5]. Password and username is set with function from Paho MQTT library: `client.username_pw_set(username, password)` in line 17. TLS security [20] is set with Paho library's function: `client.tls_set(path_to_tls_certificate, tls_version)` on line 20.

Client connects to the broker on line 23 with function `client.connect(hostname, port, keepalive)`. This function sends CONNECT packet to the cloud server with appropriate headers, such as `ClientID`, `username` and `password`. The cloud uses these headers to identify and authorize the connected device. `Keepalive` is set to 60 seconds, this interval can be set to longer, but it is not recommended to set the interval shorter. If the cloud doesn't receive any message in 2.5x time `keepalive` period, it will terminate the connection with the device. To prevent this from happening, the device will send PINGREQ control packets each `keepalive` period, if it doesn't have any other information to send.

Callbacks functions are connected to the client instance on line 26. Callback

functions are triggered when specific event happens, such as when connection is established (on\_connect callback) or when new message arrives from the cloud (on\_message callback).

```

1 def client_init(self, , func=None, *args):
2     self.client = mqtt.Client(client_id="tuyalink_"+self.
3         device_ID)
4         self.timestamp = int(time.time())
5
6     self.func = func
7     self.my_args = args
8
9     #content string - deviceId=${DeviceID},timestamp=${10-digit
10    current timestamp},secureMode=1,accessType=1
11    content_string = "deviceId="+self.device_ID+",timestamp="+
12    str(self.timestamp)+" ,secureMode=1,accessType=1"
13
14    #sha256 hash from content_string with device_secret_ID as a
15    key
16    digest = hmac.new(self.device_secret_ID.encode('UTF-8'),
17    content_string.encode('UTF-8'), hashlib.sha256)
18    signature = digest.hexdigest()
19
20    #setup username and password
21    #username string - ${DeviceID}|signMethod=hmacSha256,
22    timestamp=${10-digit current timestamp},secureMode=1,
23    accessType=1;
24    self.client.username_pw_set(self.device_ID+"|signMethod=
25    hmacSha256,timestamp="+str(self.timestamp)+" ,secureMode=1,
26    accessType=1", password=signature)
27
28    #enable TLS, set certificate
29    self.client.tls_set(ca_certs=self.cert_path, tls_version=ssl
30    .PROTOCOL_TLSv1_2)
31
32    #connect client to tuya server
33    self.client.connect('m1.tuya.cn', port=8883, keepalive
34    =60)
35
36    #connect callbacks
37    self.client.on_connect = self.on_connect
38    self.client.on_message = self.on_message
39    self.client.on_log = self.on_log

```

**Listing 5.2:** MQTT client initialization and authorization and connection

In on\_connect callback, which is triggered during connection of the client to the broker, client subscribes to properties delivery topics, so the client resubscribes these topics in case of reconnection after being disconnected from the cloud.

```

1 def on_connect(self, client, userdata, flags, rc):
2     client.subscribe("tylink/"+self.device_ID+"/thing/model/
3     get_response", qos=1)
4     client.subscribe("tylink/"+self.device_ID+"/thing/property/
5     set", qos=1)

```

**Listing 5.3:** MQTT client initialization and authorization and connection

To handle incoming messages, `on_message` callback is called, which loads the JSON content of the payload. This also triggers the `func` function, which is set during initialization of the client and is used for to perform any additional operations with the payload data. `func` function is used later when implementing gateway on the Raspberry Pi for APC power strip to send a command to the interface of the Telnet interface of the power strip after receiving a device property update from the cloud.

```

1 def on_message(self, client, userdata, msg):
2     self.incoming_message_payload = json.loads(msg.payload)
3     self.func(incoming_message_payload, *self.my_args)
4 
```

**Listing 5.4:** Callback for loading message payload from incoming messages

`report_properties(self, msgId, data)` function is used to report device properties to the cloud. `report_properties` function takes Python dictionary `data` as its input argument, which consists of pairs: "property\_name": property\_value. `properties` Python dictionary is the message body of the published message. It is constructed from unique to each message messageId (`msgId`), current timestamp and `data` dictionary. `properties` dictionary is converted to JSON payload of the message on line 9.

Properties are published into report properties topic on line 11, recommended level of QoS by Tuya is 1, QoS levels of MQTT protocol are described in chapter 3.2.

```

1 def report_properties(self, msgId, data):
2     #encapsulate data dictionary into properties dictionary
3     properties = {
4         "msgId":str(msgId),
5         "time":self.timestamp,
6         "data": data
7     }
8     #convert properties dictionary to json payload
9     payload = json.dumps(properties)
10    #publish payload
11    ret_pub = self.client.publish(str('tylink/'+self.device_ID+'
    /thing/property/report'),payload, qos = 1)

```

**Listing 5.5:** Report properties

There are two functions to start a network loop [17]. When new MQTT message is received, it is stored in the receive buffer. Loop function periodically checks the receive buffer and triggers the appropriate callback function, where the data is processed. Outcoming messages are stored in the send buffer. Loop function checks the send buffer and sends all messages it finds. Without calling the loop function, no messages can be received or sent.

`loop_start` function is non-blocking function, which starts network loop in another thread. The network loop is running until the client is disconnected or until the `loop_stop` function is called. `loop_forever` function is a blocking function for starting network loop, which runs indefinitely until the client is disconnected. As the `loop_forever` function blocks the current thread, no other commands can be executed at the time the function runs, including sending messages.

## 5.4 Using the Tuya MQTT client module

In this section, I will explain, how to use the Tuya MQTT client module in a Python code integrate the device into Tuya ecosystem. This will be shown on a simple example of a device, that has two properties: `on_off_state` of boolean data type and `brightness` of integer data type.

The module `tuya_mqtt_client` is imported on line 1. The instance of the `TuyaClient` class is created and initialized with `DeviceID`, `DeviceSecretID` and path to TLS certificate on line 7. `client_init(func)` function is called to connect and authenticate the client to the Tuya cloud on line 8. `print_property` function is passed in the `client_init` function, which will process the payload of any incoming messages (in this example print the payload).

To report status of the device to the cloud, `report_properties("messageID", data_dictionary)` is called, where `data_dictionary` is Python dictionary containing names and values of properties to be reported, as seen on line 15.

To start loop for handling incoming messages, `loop_forever()` function is called as seen on line 17. Alternatively, `loop_start()` and `loop_stop()` functions can be used to create a finite loop and keep the current thread free for other operations.

```

1 import tuya_mqtt_client
2
3 def print_property_update(payload):
4     print(payload)
5
6 if __name__ == '__main__':
7     client = tuya_mqtt_client.TuyaClient("DeviceID", "
8         DeviceSecretID", "path_to_TLS_certificate")
9     client.client_init(print_property_update)
10
11     data_dict = {
12         "on_off_state": True,
13         "brightness": 11
14     }
15     client.report_properties("123", data_dict)
16
17     client.loop_forever()

```

**Listing 5.6:** Example code for using the Tuya MQTT client module





## Chapter 6

# Setting up a new device on the Tuya cloud platform

## 6.1 Setting up a new device on Tuya cloud

### 6.1.1 Introduction

In this chapter, I will describe how to configure a new device on the Tuya cloud and various settings on the cloud side of the device integration.

Product represents a group of physical devices with same properties and capabilities. Grouping devices of same types to products makes management and configuration of devices on Tuya cloud scalable for manufacturers, who produce larger series of devices of the same type.

Tuya's monetization model is based on paying for licenses for larger series of devices for the same product. Every device that connects to Tuya cloud must be issued a license, which contains unique identifier such as DeviceID and DeviceSecretID, which are necessary to authenticate the device to Tuya cloud. For every product, Tuya issues a limited number of free licenses for debugging purposes for data center in China, developers outside chinese region must pay for the licenses. Manufacturers of smart Tuya devices buy and manage those licenses in bulk.

### 6.1.2 Creating a product

Configuration of a device on Tuya cloud is done through Tuya IoT Development platform (<https://iot.tuya.com/>). Upon entering this page, you will be asked to login or register to your Tuya account. To access product development menu, click on "Product", then on "Product Development" on left side of the screen as shown on [Figure 6.1]. To create a new product, click on the "Create" button on the right side of the screen. In the centre of the screen is a list of all products on this account. To change configuration of a product, click on the "Develop" button on the right side of the screen.

## 6. Setting up a new device on the Tuya cloud platform

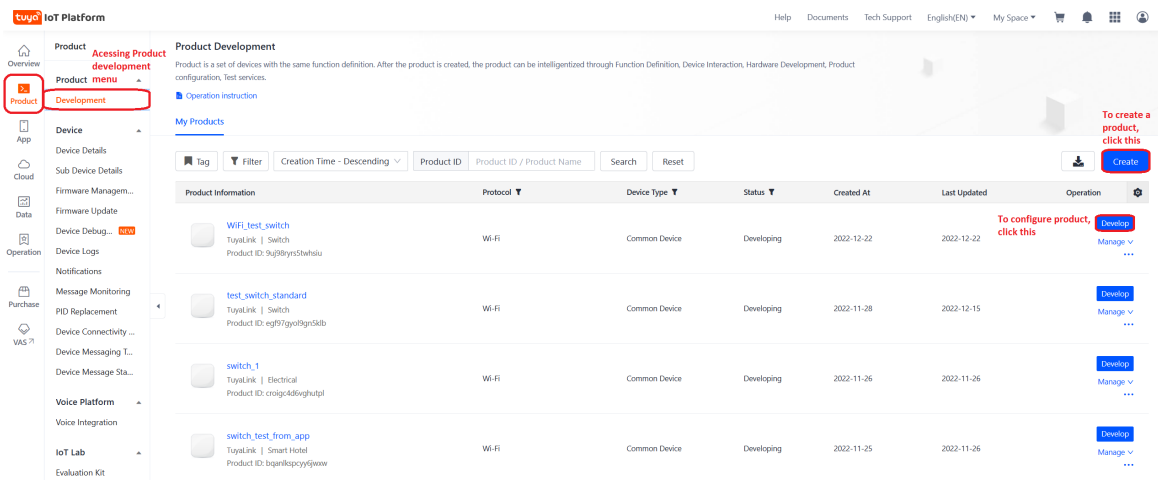


Figure 6.1: Product development menu on Tuya IoT platform

### Device type

After clicking on the "Create" button, the device type needs to be selected as shown on [Figure 6.2]. There are many types of devices to select from, such as power strips, sockets, switches, various types of sensors, lights, cameras and others. If desired device category cannot be found, or if you want to configure device from scratch, option "Can't find the category?" can be chosen, located at the bottom left of the menu [Figure 6.2]. It is not critical to choose device category, as device must be configured the same way in later steps regardless.

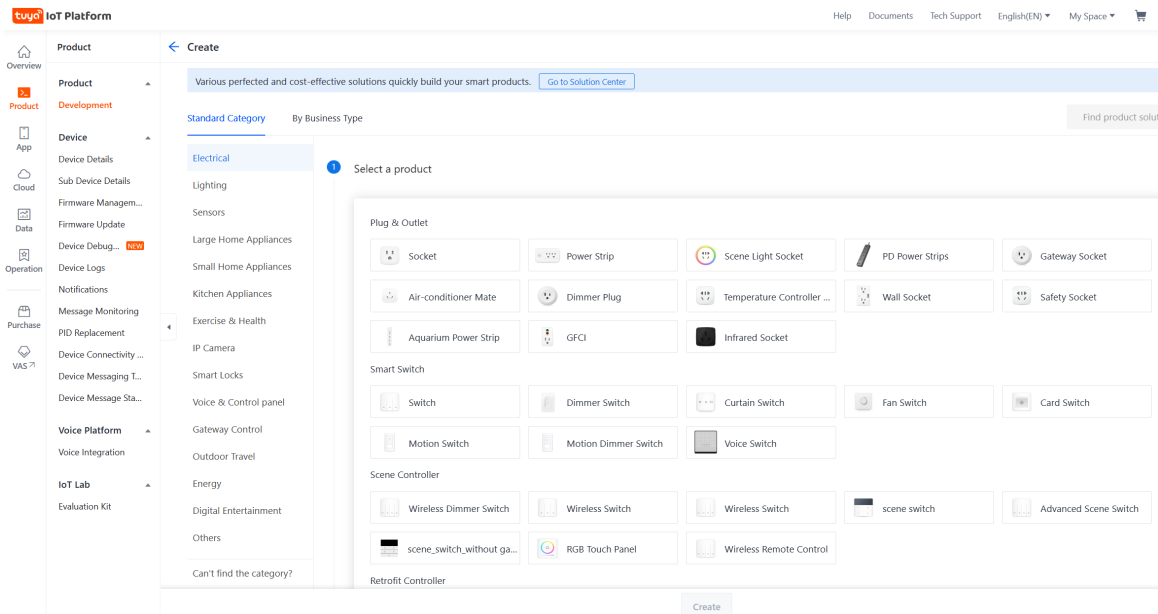


Figure 6.2: Device type selection screen

## TuyaOS and TuyaLink development methods

Next step in configuring product on Tuya is choosing between TuyaOS and TuyaLink development methods, as shown in [Figure 6.3].

TuyaOS is an operating system based on RTOS, Linux and Non-OS [22]. Tuya sells various microcontroller boards, network modules, or sandwich modules (which can be stacked on popular types of microcontrollers, such as Arduino, or Nucleo to give these microcontrollers connectivity with Tuya), that can be flashed with TuyaOS firmware. Development for TuyaOS is done through Wind IDE, which handles compilation and flashing of firmware to the board.

Second method of development is TuyaLink, which uses MQTT protocol to connect devices to Tuya cloud. TuyaLink method defines MQTT topics, authentication with DeviceID and DeviceSecretID identifiers and defined format of message payload to allow any device, that supports MQTT protocol and has network connectivity to be able to connect to Tuya cloud. In this thesis, the main focus will be on TuyaLink method of connectivity.

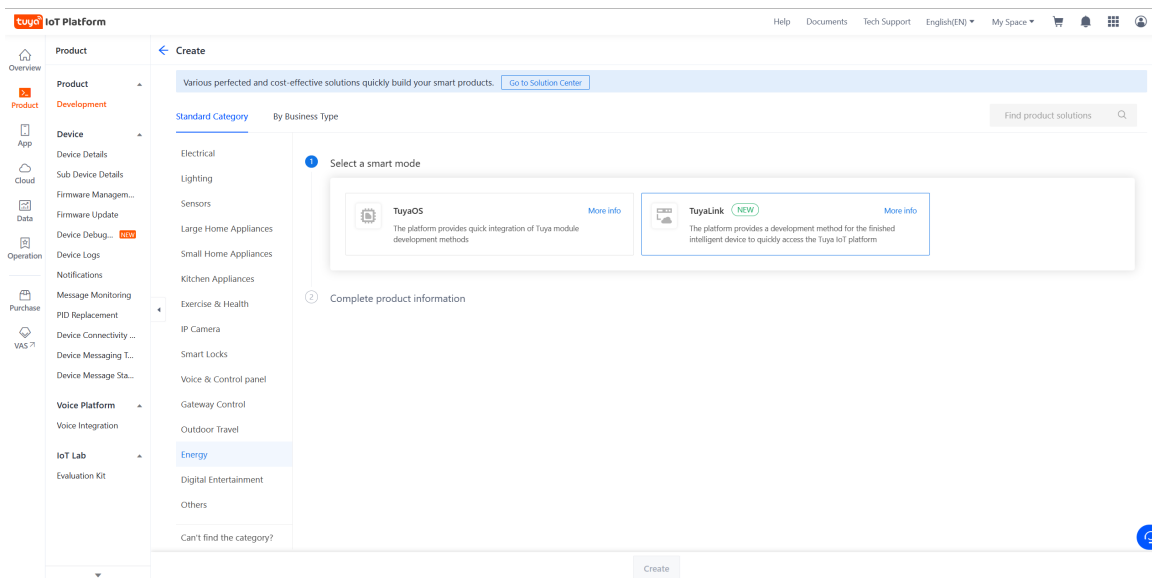


Figure 6.3: Development method selection screen

## Product information

After choosing a development method, product information must be filled out and communication protocol, data protocol, device type must be selected, setup screen is shown on [Figure 6.4].

Device type checkbox is used to select the role of device in the topology, when connecting the device to the Tuya cloud. Common device option means, that device is directly connected to the Tuya cloud via router. Gateway device manages topological relationships with sub-devices and aggregates messages from subdevices and sends them to the cloud. Gateway device can

report data from multiple sub-devices in bulk in one message. Sub-device must be connected to the cloud through gateway device.

Data protocol checkbox is used to select the format of the payload of MQTT messages. Tuya Standard protocol option means, that format of the payload of the MQTT messages will be JSON, structured as specified by Tuya.

When using *Standard protocol* option, device must properly structure the message payload into JSON format. When using *Custom* option, raw byte data are received by the cloud, and data parsing script must be implemented by the developer and uploaded to the cloud.

Protocol checkbox is used to select the communication protocol for the northbound interface of the device, the most popular protocols are present (Wi-Fi, Bluetooth, Ethernet).

2 Complete product information

\* Product Name: Enter product name

Product Model: Enter your product model, separated by commas

\* Product Description: Briefly describe product functions and application scenarios to help Tuya recommend the best applicable solutions.

\* Device Type: Common Device Gateway Device Gateway Sub-Device  
Refers to the device that can be directly connected or connected to the Internet of Things platform through a router, or can be mounted as a sub-device under the gateway, but cannot be mounted to a sub-device.

\* Data Protocol: Tuya Standard Protocol Custom  
Refers to the use of the JSON device model defined by the Tuya standard.

\* Protocol:  Wi-Fi  Wi-Fi and Bluetooth Combo  Bluetooth Mesh  Bluetooth  NB-IoT  Ethernet  
 2G/3G/4G  5G  other

Figure 6.4: Product information configuration screen

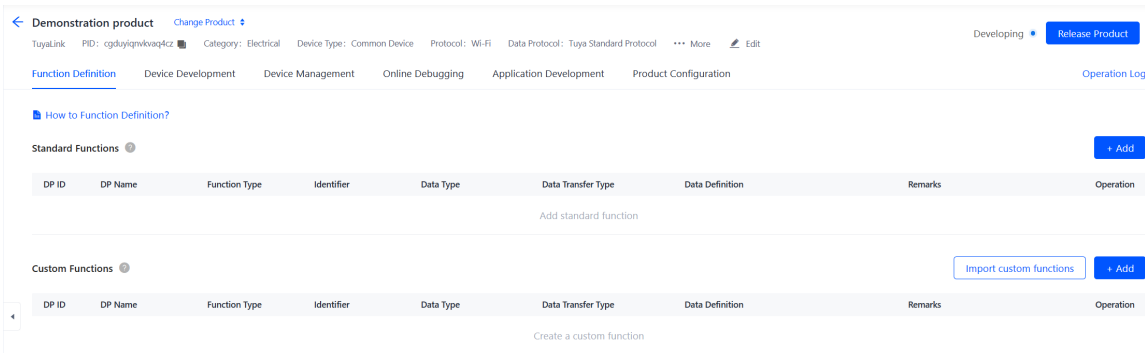
After finishing the configuration in the Product information menu, click the "create" button on the bottom part of the screen to create the product.

## 6.2 Product development menu

After clicking on the create button on the previous step, product development menu will show up. Product development menu is used to set up the model of a product, get licenses and identifiers for devices, use online debugging to verify cloud to device connectivity, manage devices, set up application panels for devices and bind devices to mobile applications. In the following sections, I will describe the sections of the product development menu and how to use it to set up a new device.

### 6.2.1 Function definition

Function definition menu is used to configure a device model (chapter 4) on Tuya cloud. *Properties*, *actions* and *events* are together called *functions* of the device model. Properties, actions and events are used to model the device and are exchanged during communication between the Tuya cloud and a device. To access function definition menu click on the "Function Definition" on the top bar, as shown in [Figure 6.5].



**Figure 6.5:** Function definition

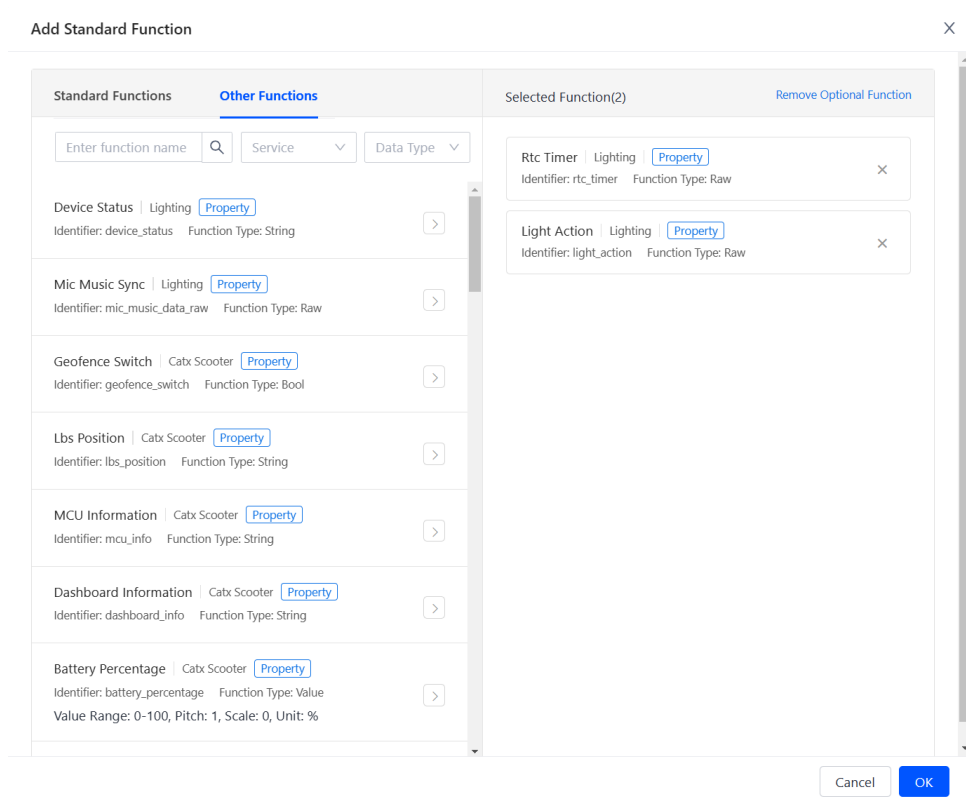
In the middle of the function definition screen, there are two lists of functions: *Standard functions* and *Custom functions*. Standard functions refer to a library of predefined functions, that are preconfigured by Tuya for various functions of many types of devices. Custom functions are created by developer from scratch.

To add a Standard function, click on the blue "Add" button on the right side of the standard function list, then choose any number of Standard functions from the left side of the pop-up menu [Figure 6.6], selected functions will be shown on the right side of the pop-up menu, click blue "OK" button to add all selected Standard functions.

To add a Custom function, click on the blue "Add" button on the right side of the Custom function list. Alternatively, Custom function definitions can be imported in bulk from .xlsx Excel spreadsheet file by clicking the "Import custom function button". After clicking the "Add" button, Create custom function menu will appear on the right side of the screen [Figure 6.7].

Every function must have unique DP ID (Data point id), which must be numerical value between 101 and 499. In the "Function name" box, fill out the name of the function, this can be any string.

Identifier is as string used in JSON payload of MQTT messages, which are being sent between the cloud and a device, to identify properties, actions and events (described in chapter 4). Identifier must be string made of uppercase and lowercase letters, numbers and underscores (for example: Switch\_1). When constructing JSON payload on the device side, it is important to use



**Figure 6.6:** Standard functions pop-up menu

the same identifiers, as are configured on the cloud side, otherwise the cloud cannot recognize the reported data.

In the "Function Type" checkbox, select the desired type of function. There are three types of functions: properties, actions and events. These are used to model physical device and are being sent during communication between the Tuya cloud and a device. Device model and properties, actions and events are described in more detail in chapter 4.

When property is selected as a function type, next step is to select the data type of the property. Properties can have various data types, such as numerical value, float, string, structure, array and others.

Next step when configuring a property is to select data transfer type in the last checkbox of this menu. There are three data transfer types available: send-only (property values can be only sent from the cloud to the device), report-only (property values can be only sent from the device to the cloud), or send and report (property values can be sent both ways).

When configuring an event, configure the DP ID, the function name and the identifier the same way as when configuring a Property and choose Event in the function type checkbox. Events are report-only notifications to the cloud from the device. As an option, *output parameters* can be added to the event, by clicking the "Add parameter button", then fill out the parameter name, identifier and select the data type [Figure 6.8]. Output parameters

**Create Custom Function**
✕

① You can build a product equipment model by focusing on the "properties" features of the product, the "events" that need to be monitored, and the complex "actions" that can be performed according to the custom function. [How to Function Definition?](#)

DP ID:

\* Function Name ⓘ:

\* Identifier ⓘ:

\* Function Type ⓘ: Property Event Action

\* Data Type :

\* Value Range:  ~

\* Pitch:

\* Scale ⓘ:

Unit:

\* Data Transfer Type ⓘ:  Send and Report  Report Only  
 Send Only

Remark:

**Figure 6.7:** Creating a custom function

can be used to provide additional information (variable value) to be sent alongside the event message to the cloud.

Actions are request-response 2-way messages, which instruct the device to perform a specific task. For actions, both *input and output parameters* can be configured. The configuration process for parameters of actions is the same as for parameters of events, as described in the previous paragraph.

To finish configuration of a function, and add the function to the product, click on the "Ok" button on the bottom right of the Create custom function menu.

**Add Parameter**
✕

\* Parameter Name:

\* Identifier:

\* Data Type:

Maximum Length:  Byte

**Figure 6.8:** Adding parameter to the Event or Action

## 6.2.2 Device development menu

Second section of the product development menu is the device development menu. Here we can select development method.

There are three development methods available: Open protocol, Cloud-to-Cloud integration and Edge gateway sub-device [Figure 6.9]. Open protocol is the option I will be focusing on.

Open protocol option uses MQTT protocol to connect directly connected devices, gateway devices and gateway sub-devices to the MQTT gateway on the cloud side, as shown on diagram in [Figure 6.9].

Cloud-to-Cloud integration option uses APIs to connect to devices indirectly through third-party clouds.

Edge gateway sub-device means the cloud uses edge gateway to offload part of the workload from the cloud and communicate with sub-devices connected to the edge gateway. Edge gateway run on the hardware in standard container [23]. All services are installed on edge gateway from docker images.

**Figure 6.9:** Choosing development method



### 6.2.3 Device management

Device management section is used to manage licenses for devices. After assigning a license to the device, DeviceID and DeviceSecretID identifiers are obtained, which are used during authentication process of the device to the cloud at the start of the MQTT communication. At the center of the screen all devices, that are registered under the product at the same data center, can be seen [Figure 6.10]. Data center can be selected from the drop-down list at the upper left part of the screen. Correct data center for the region must be selected, otherwise Tuya won't respond to the MQTT messages. Tuya provides 6 free licenses for devices for chinese data center, however these unfortunately cannot be used outside China. To connect devices to Tuya outside China, licenses for the correct region must be purchased, they did cost me 20\$ for 10 device licenses.

The screenshot displays the 'Device Management' interface for a 'Demonstration product'. At the top, there are navigation tabs: 'Function Definition', 'Device Development', 'Device Management' (active), 'Online Debugging', 'Application Development', and 'Product Configuration'. Below the navigation, there's a 'How to Device Management?' section. The 'Device List' section features a dropdown menu set to 'China Data Center' and a 'Remaining Licenses: 0pcs' indicator. A search bar includes a 'Binding...' dropdown, a 'Time range' selector with 'Start date' and 'End date' fields, and an 'Enter: Device ID/Registration ID' field with 'Search' and 'Reset' buttons. A 'Download QR Code' button is on the left, and 'Batch Management' and 'Register Device' buttons are on the right. The table below has columns: Device name/Device ID, Device Type, Device Status, RegistrationID, Registration Time, Activation Time, Remark, and Operation. One device is listed with details: 'Demonstration product743d 26120f0b-4541839591e04c', 'Common Device', 'Unbound offline', '7Ht8v2RkXfWhU8SGDg', '2023-02-03 13:55:24', '2023-02-03 13:55:24', and a 'Register Device' icon. The bottom right shows 'Total 1 Items' and navigation arrows.

Figure 6.10: Device management menu

To register a new device click on the "Register device" button on the right side of the screen. After registering a device a pop-up menu will appear [Figure 6.11]. At the top of the menu is shown which product and data center device belongs to and number of unused licenses. If there are no remaining licenses left, it is possible to reassign a license from other device by clicking on the "Assign licenses" text next to the number of remaining licenses.

There are three registration methods available: Single registration, Batch import and Auto-registration. To register a single device, choose Single registration at the registration method checkbox and fill out the RegistrationID. RegistrationID is a string made from numbers and letters, which must be unique to each device registered under the same product. Click "OK" button to confirm registration.

To import multiple devices at the same time from .xlsx Microsoft Excel spreadsheet, select Batch Import. Up to 1000 devices can be registered at once. In the second column of the Excel spreadsheet. After filling out the spreadsheet, upload it by clicking on the "Upload attachment".

To register multiple devices at once automatically, Auto-registration can be used. In the "Device Quantity" field, fill out the number of devices to be added, this number must not exceed the number of available licenses. When

using Auto-registration, Tuya automatically registers, assigns licenses and generates and assigns RegistrationID's to the selected number of devices. Click "OK" button to confirm registration.

The screenshot shows a 'Register Device' dialog box with the following content:

- Warning:** You need to purchase and assign the authorization code to the current product, and the device authorization certificate can be generated through device registration, which can be used for device access. To register other data center devices, please go to the current product development process-device management page and register.
- Product:** Belongs to the Product: Demonstration product
- License:** Remaining license: 0pcs [Assign License](#)
- Data Center:** data center: China Data Center
- Registration Method:**  Single registration  Batch Import  Auto-registration
- RegistrationID:** The unique identification code of the device under the product, if not filled ...
- Remark:** Fill in the remarks
- Buttons:** Cancel, OK

Figure 6.11: Registering a single device

#### 6.2.4 Online debugging

This section of the product development menu is used to debugging and verifying connectivity between devices and the cloud. We can view message logs, debug reporting and received messages with property updates or response messages to actions [Figure 6.12]. In the upper left part of the Online debugging menu, we can select data center, select device to be debugged, which will be listed by its DeviceID, and we can see the status of the device (device is online or offline). In the left part of the Online debugging menu, properties and actions can be debugged. In the right part of the Online debugging menu, we can see real-time log of all messages being sent between the debugged device and the cloud.

To debug properties, select "Property debugging" in the left part of the screen. List of all properties of the device will show up. Clicking on "Get" will write out the last value of the property received by the cloud to the "Parameter value" box. If no property value was reported, this will write out the default value of the property. Clicking on "Set" will generate and send MQTT message to the debugged device with property value filled out in the "Parameter value".

To debug actions, select "Action control" in the left part of the screen. Then select one of the actions from drop-down list, as seen on [Figure 6.13]. List of all input parameters of the selected action will be displayed, along with their data types. In the "Parameter value", fill out the value of the input parameters. To send the instruction message to the target device, click on

The screenshot displays the 'Online Debugging' menu for a 'Demonstration product'. At the top, there are navigation tabs: Function Definition, Device Development, Device Management, **Online Debugging**, Application Development, Product Configuration, and Operation Log. Below the tabs, there's a header with product details: TuyaLink, PID: qjbtg27sydwkap11, Category: Electrical, Device Type: Common Device, Protocol: Wi-Fi, Data Protocol: Tuya Standard Protocol, and a 'Release Product' button. The main content area is divided into two sections. On the left, 'Device Information' shows 'China Data Center' and '26120fdb4541839591e0qc' with a 'Switch device' dropdown and a 'Device online' status. Below this is a 'Property Debugging' section with a table for parameter management. On the right, the 'Real-Time Log' section has a 'Log type' dropdown, 'Clear Log', 'Manual Refresh', and 'Auto Refresh' (checked) buttons. The log table shows three entries: a device property delivery with a JSON payload, a device online status, and a device offline status.

Parameter name	Parameter value	Operation
prop1	Select parameter value	Get Set

Time/Log Type	Content
2023-02-08 23:02:55 Device property delivery	{"data": {"prop_1": false}, "msgId": "855213252444950529", "time": "1675890175"}
2023-02-08 23:02:35 Device online	
2023-02-08 23:02:33 Device offline	closed

**Figure 6.12:** Online debugging menu

the "Send instructions" button.

As events are report-only notifications from the device to the cloud, there is no debugging menu for the events, but all incoming event messages should be shown on the real-time log.

The real-time log on the right side of the screen shows every incoming and outgoing message, that cloud receives from the debugged device or sends to the debugged device. The log lists timestamp and the log type on the left side and content of the message on the right side. Incoming messages from the device to the cloud are marked with "report" keyword in log type (device property reporting, device action report and device event report log types). Outgoing messages from the cloud to the device are marked with "delivery" keyword in log type (device property delivery and device action delivery log types). Content field of each log shows JSON formatted payload of the MQTT messages being sent between the cloud and the device. Device online and Device offline logs are used to indicate the status of connection between the device and the cloud.

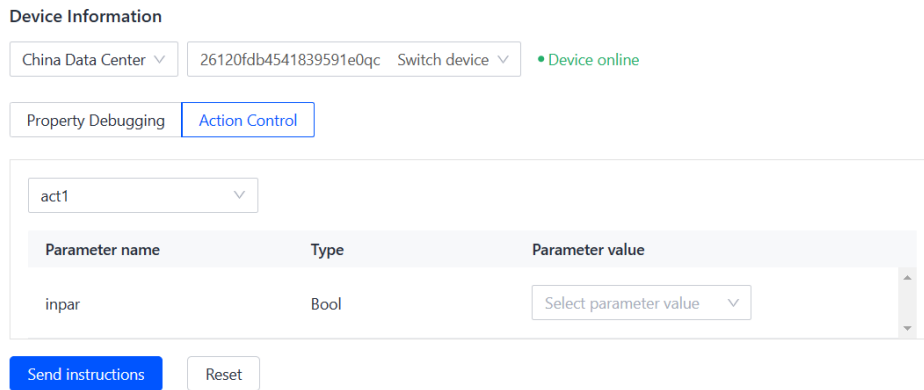


Figure 6.13: Debugging Actions

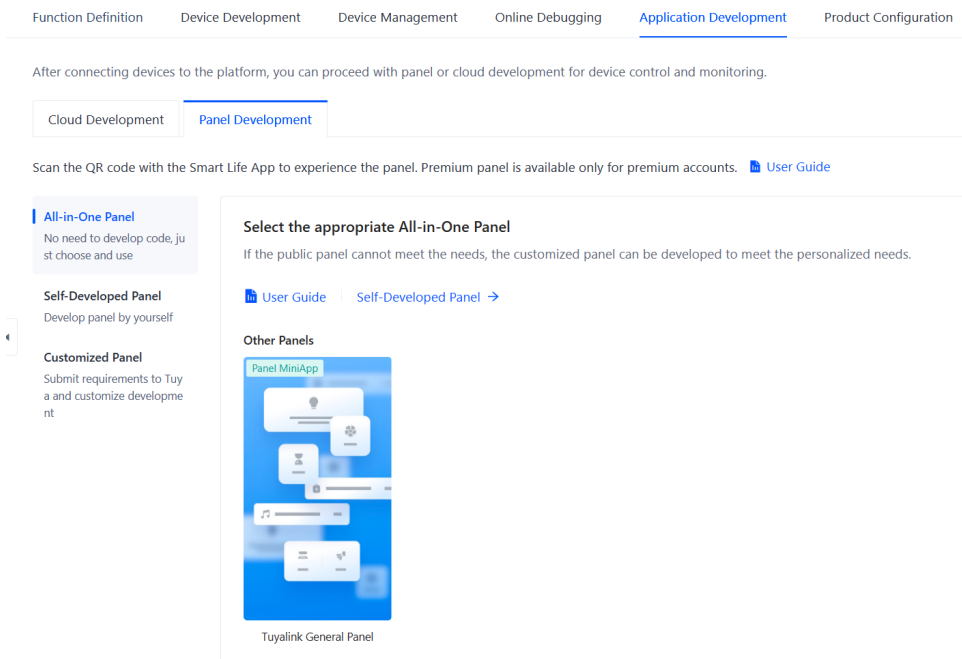
### 6.2.5 Application development

In application development, section, interfaces for monitoring and controlling the devices can be configured. In application development section there are two tabs: Cloud development and Panel development.

When creating web applications for controlling and monitoring a device, cloud development is used to configure cloud services, in order for the web application to connect to the APIs of the Tuya cloud. Cloud development tab of the Application development menu provides a brief overview of steps of the cloud development process along with links to the menus, where the configuration is done. Cloud development is mainly done through cloud development menu, which will be described in detail in chapter 8.

In Panel development tab, we can select control panel to be used for this device in the Tuya Smart Life App. To select a panel, click on the All-in-One panel tab on the left [Figure 6.14]. Here you should see a list of available panels. By default, there will be one default panel available, named as "Tuyalink general panel". This panel is automatically generated from the set of properties, actions and events, that we have configured for the device and provides basic functionality, such as displaying and changing Property values.

To create our own panel, click on the Self-Developed panel tab on the left. Here we can create our custom panel by clicking on "Create Smart Mini App Panel" button, which will redirect us to Smart MiniApp developer platform, which is environment for developing panels for Tuya Smart Life App. This feature is sadly locked behind paid subscription.



**Figure 6.14:** Panel development tab of the Application development menu

## 6.2.6 Product configuration

Last section of the Product development menu contains a few additional settings for binding devices to the mobile applications, multi language support, enabling scenarios and setting up the data parsing script for custom data protocol of the payload of the MQTT messages.

### Device binding configuration

In Device binding configuration settings we can select which accounts will have permission to bind the smart devices to the mobile application and the cloud project. Cloud project is a unit of data storage on the Tuya IoT Development platform and is used to manage and configure the associated device permissions, API permissions and data assets [7]. Cloud projects are configured through cloud development menu. To change device binding permission scope, click on settings button in the binding configuration box. In default, option "Current account" is selected, which grants permission to bind devices to the cloud project only to the currently logged in account. To grant this permission to all IoT Platform accounts and all Tuya App accounts, select "All IoT Platform accounts" option.

### Multi-language management

In Multi-language management settings, we can add translations of names of products, functions and other terms to ensure proper localization of the Tuya Smart Life app in the desired language.

## Scenario connection settings

In scenario connection settings, we can configure which properties are configurable by user in the Tuya Smart Life application for the automation scenarios [19]. After clicking on the settings button, the settings menu will show up, which is shown on [Figure 6.15]. The properties that are configured in the Function definition menu will be listed under two categories: Scenario trigger condition settings and Scenario implementation task settings, some of them might be listed under both.

Scenario trigger conditions are properties that are used to trigger automation scenarios. Scenario implementation tasks are properties changed on reaction to trigger conditions. Automation scenarios are used to let one device act based on input from other device. For example user could have a smart thermometer and a smart air conditioner. User than could configure in his Tuya Smart application an automation scenario like this: when the temperature sensed by the thermometer rises above a certain value, the air conditioner turns on. All properties are in default enabled for users to use for automation scenarios. If we want to prevent user to use a property of our device in automation scenario, untick the checkbox next to the Property.

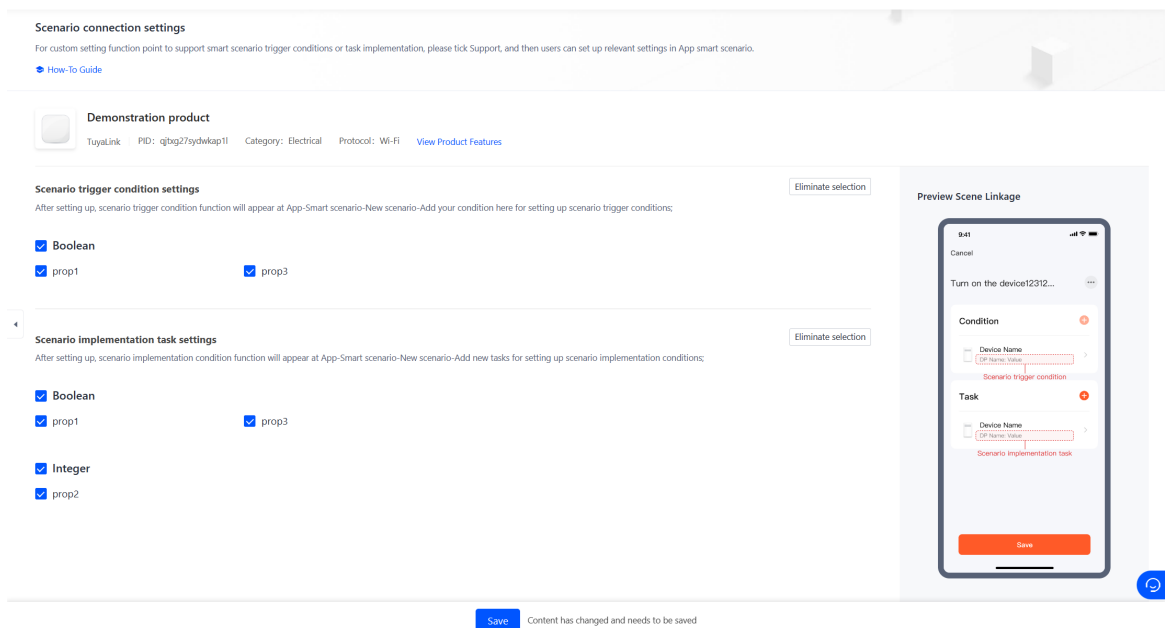


Figure 6.15: Scenario connection settings

## Remote configuration

In remote configuration settings JSON configuration file can be prepared, which can be requested by the device. Device can use the configuration file for any kind of configuration, parsing must be implemented on the side of the device. Device can request a configuration file by sending a configuration request MQTT message to the `tylink/{deviceId}/ext/config/get` topic [18].

## Data parsing

This section of the product development menu is shown only, if custom protocol was chosen as a data protocol when creating a product, as described in chapter 6.1.2. If a device cannot structure the payload of the MQTT messages to the JSON format, it can send raw data to the cloud. Developer than must implement script that will handle the conversion between raw data and TuyaLink JSON format [4].

To create the script for data conversion, click on the "Data parsing" tab on the upper bar of the Product development menu, then click on the "Script debugging" button. In the centre of the screen, there is a prepared prototype for the conversion script [Figure 6.16]. Two functions must be implemented in the data conversion script: function `rawDataToTyLink(rawData) {}`, which converts incoming raw data to the TuyaLink standard JSON format, so the cloud can understand them, and function `tyLinkToRawData(tylinkData) {}`, which converts outgoing JSON formatted data to raw data. Currently, only supported language for this conversion script is JavaScript. At the bottom of the screen, there is an option to simulate the communication to verify the proper parsing. After the script is done, click on the "Release as official version" button to make the script active for this project.

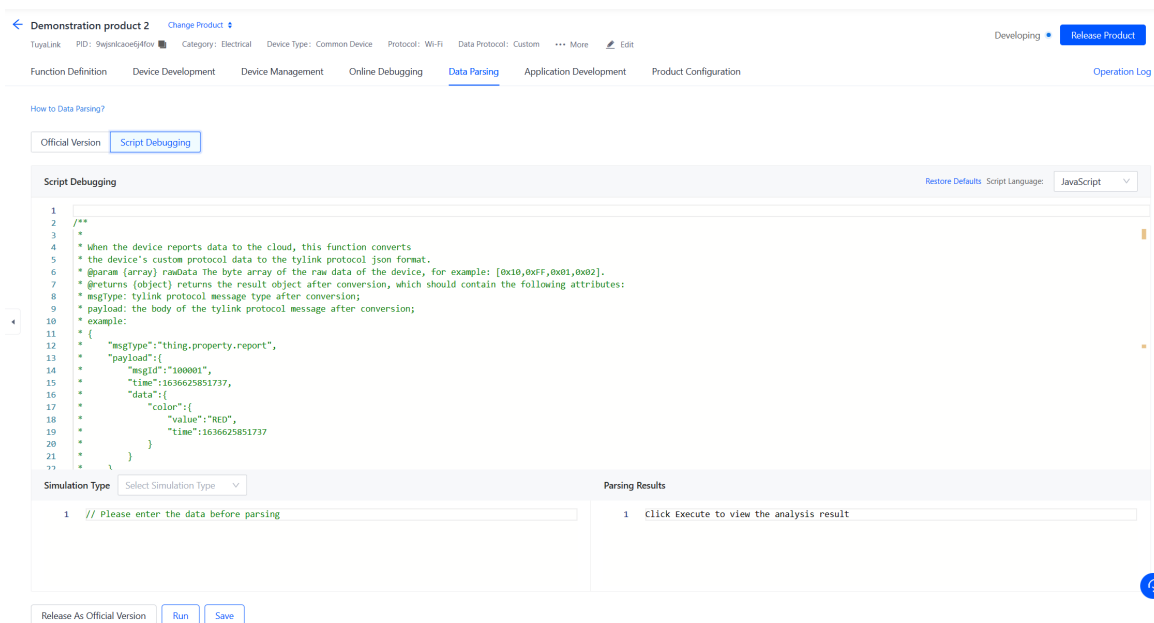


Figure 6.16: Data parsing menu

## 6.2.7 Finishing product configuration

After all the previously described configuration has been done, and the connectivity between the cloud and the device was verified with the Online Debugging tool, as described in chapter 6.2.4, it is time to release the Product by clicking on the blue "Release Product" button in the top right corner of





## Chapter 7

# Developing a gateway for APC Telnet power strip integration.

### 7.1 Introduction

In this chapter, I will be describing how I implemented gateway for controlling the AP7920 rack PDU (power distribution unit), using Telnet protocol for communication between the power strip and the gateway device and MQTT protocol for communication between the gateway device and Tuya cloud. PDU used is manufactured by APC, has 8 power outlets and supports Telnet protocol for remote switching and management of power outlets. Raspberry Pi 3 model B will be used as the hardware for the gateway. Raspberry Pi is a small single board computer (SBC). Raspberry Pi is popular platform for IoT project, because it is compact, has Ethernet port and Wi-Fi connectivity and has many interfaces, such as USB, I2C, SPI or digital I/O pins, so non-standard peripherals can be connected.

### 7.2 Telnet interface of the power strip

Telnet is an application layer protocol for that provides remote access to virtual terminals of remote systems [26]. Basically that means, that we are able to remotely access command line of a computer. Telnet is used to connect to TCP port 23, on which the server (in this case the PDU) is listening.

The Telnet interface is protected by username and password. The Telnet interface has tree structure numbered options on each level, example is shown on listing 7.1. To select an option, number of the option is typed and confirmed with Enter key. To move back up one level, Escape character is used. To send commands via the Telnet interface, I must emulate navigating through the tree menu and selecting options with key presses.

```
1 American Power Conversion      Network Management Card AOS v2.6.4
2 (c) Copyright 2004 All Rights Reserved Rack PDU APP          v2.6.5
3 -----
4 Name       : RackPDU                      Date : 05/17/2023
5 Contact    : Unknown                      Time : 21:29:11
6 Location   : Unknown                      User  : Administrator
```



```

1 def property_handler(payload):
2     try:
3         loop = asyncio.get_event_loop()
4     except RuntimeError as e:
5         if str(e).startswith('There is no current event loop in
thread'):
6             loop = asyncio.new_event_loop()
7             asyncio.set_event_loop(loop)
8         else:
9             raise
10
11     global outlet_number
12
13     for key in payload["data"]:
14         outlet_code = key
15
16
17     switch = {
18         "sw1": "1",
19         "sw2": "2",
20         "sw3": "3",
21         "sw4": "4",
22         "sw5": "5",
23         "sw6": "6",
24         "sw7": "7",
25         "sw8": "8"
26     }
27
28     #print(switch[outlet_code])
29     outlet_number = switch[outlet_code]
30
31     if (payload["data"][outlet_code] == True):
32         sw8 = True
33         coro = telnetlib3.open_connection(HOST, 23, shell=
shell_ON)
34     elif (payload["data"][outlet_code] == False):
35         sw8 = False
36         coro = telnetlib3.open_connection(HOST, 23, shell=
shell_OFF)
37     reader, writer = loop.run_until_complete(coro)
38     loop.run_until_complete(writer.protocol.waiter_closed)

```

Listing 7.3: property\_handler function

Coroutines for sending commands to the telnet interface call number of other functions, which are used to move up and down through the tree structure of the menu and select the options to change. For convenience, I have written several functions, which are used for login, logout, getting to the outlet management level of the tree menu, switching the power outlet on/off or logout. They all essentially only use three simple methods of the telnetlib3 library.

*reader.readuntil* method is used to parse the text output until user's input is expected. *writer.write* method is used to send characters to the telnet interface of the PDU, which is used for moving up and down the levels of the tree structured menu and selecting options. *writer.drain* method is used to

flush the writer's buffer.

To get the status of a device, *report\_status* function is used, which parses the output from the Telnet interface of the power outlet, extracts the status of the device using regex (regular expressions) and stores it into *reported\_data* variable.

```
1 async def get_status(reader, writer):
2     await tn_login(reader, writer)
3     await tn_device_manager(reader, writer)
4     header_data = await tn_device_control(reader, writer)
5
6     status_list = re.findall("ON|OFF", header_data)
7
8     global reported_data
9     switch = {
10        "ON": True,
11        "OFF": False
12    }
13    reported_data = {
14        "sw1": switch[status_list[0]],
15        "sw2": switch[status_list[1]],
16        "sw3": switch[status_list[2]],
17        "sw4": switch[status_list[3]],
18        "sw5": switch[status_list[4]],
19        "sw6": switch[status_list[5]],
20        "sw7": switch[status_list[6]],
21        "sw8": switch[status_list[7]],
22    }
23
24    await tn_escape(reader, writer)
25    await tn_escape(reader, writer)
26    await tn_escape(reader, writer)
27    await tn_escape(reader, writer)
28
29    await tn_logout(reader, writer)
```

**Listing 7.4:** *report\_status* function

## Chapter 8

### Cloud development

#### 8.1 Introduction

Cloud development menu of the Tuya development platform is used for management and authorization of API services, API debugging, third-party cloud integration setup and granting permissions to users. Authorization keys for the API can be found here. To use Tuya's APIs to get status of devices and control devices, API services must be enabled in the Cloud development menu. The only way to control and monitor devices without changing any settings in the Cloud development menu is through Tuya Smart Life app, which can be paired to the device in the Product development menu through QR code. In the following chapters, I will describe the sections of the Cloud development and how to set them up to receive and respond to API calls. This will be later used to create a web interface for controlling and monitoring of telnet power strip.

#### 8.2 Cloud project setup

To access Cloud development menu, click on the "Cloud" button on the vertical bar on the left side of the screen and then click on the "Development" button on the vertical bar, that will appear right next to it [Figure 8.1]. As a next step, cloud project needs to be created. Cloud project is collection of resources on Tuya cloud. Various device permissions, API permissions and data assets are configured and managed based on the cloud projects [7]. Resources deployed for each project are isolated from those for other projects.

To create a new cloud project, click on the blue "Create Cloud Project" button on the upper right part of the screen [Fig. 8.1]. Pop-up menu will appear where you fill out name of the project, description, industry, development method and data center. It is very important to select the correct data center for your region, as Tuya does not support cross-region connectivity between the devices, the cloud and the apps.

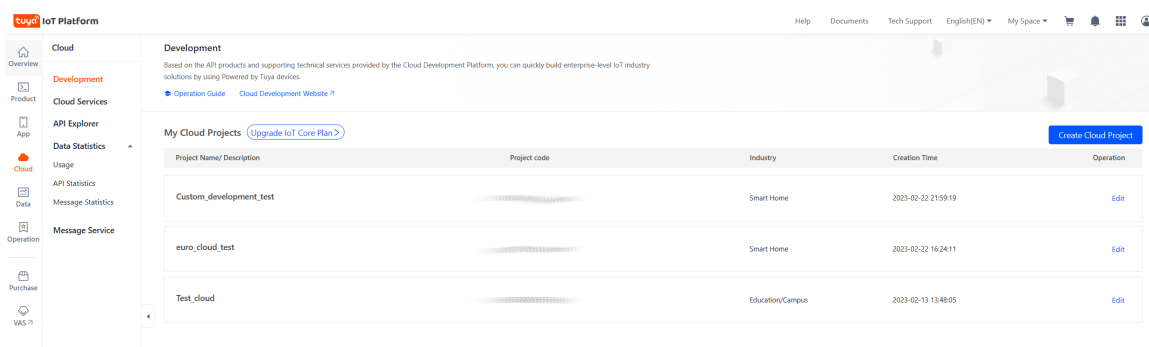


Figure 8.1: Cloud development menu

## 8.3 Sections of the Cloud development menu

### 8.3.1 Overview

In "Overview" tab, we can find two important things: authorization keys and cloud authorization IP allowlist toggle switch. Authorization keys (Client ID and Client secret) are used during the process of generating the signature for the API requests.

Toggleing on the Cloud authorization IP allowlist allows you to set an IP whitelist of IP addresses allowed to access this cloud project's resources (such as linked devices) through APIs. All requests from IP addresses, that are not on the whitelist will be rejected. This is purely optional security feature.

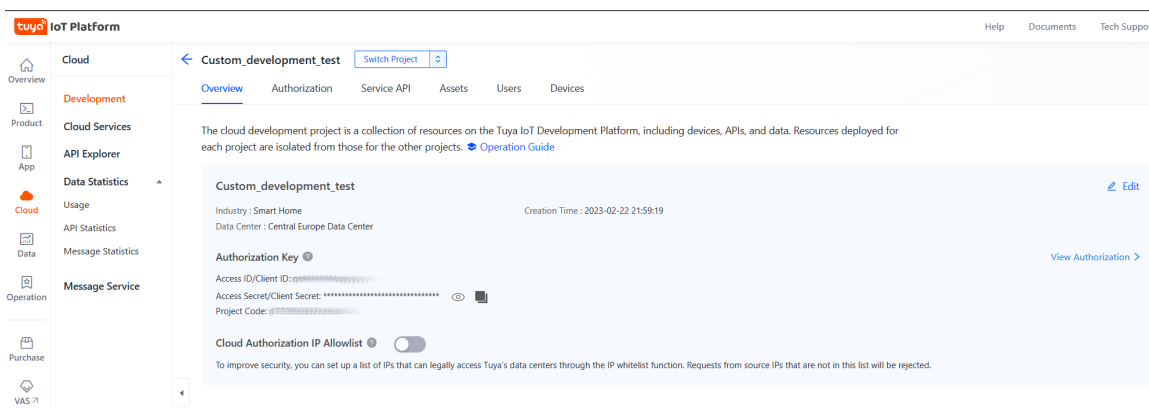


Figure 8.2: Overview tab of the cloud development menu

### 8.3.2 Authorization

Authorization tab of the Cloud development menu allows you to view and manage various authorization keys used for accessing cloud project's data and resources.

Under the "Cloud authorization" tab are by default displayed authorization keys used during the process of generating the signature for the API requests.

It is important to keep them safe, as anyone with knowledge of the authorization keys (Client ID and Client secret) could potentially access your cloud's project resources (such as devices linked to this cloud project).

Under the "App authorization" tab, authorization for custom mobile applications (other than standard Tuya Smart Life application) can be added. These applications can be based on Tuya App SDK for iOS or Android, or completely custom. Bundle identifier (unique ID for each app iOS or Android) must be filled in when creating a new App authorization, new set of authorization keys is generated for each application.

Under "WeChat Mini Program" tab, authorization for Mini Program (sub-application in chinese WeChat messaging application) can be added.

In "Third-party authorization" tab, authorization for another cloud project to access resources of this cloud project can be granted. Use project number, which can be found on the "Overview" tab of each cloud project to select which project grant the authorization and select the specific APIs to be authorized.

### ■ 8.3.3 Service API

In service API section of the Cloud development menu, we can select, which API to authorize for this cloud project. Only authorized APIs will respond to API requests. There are many APIs, that can be authorized to add functionality to the cloud project, I will briefly cover a few, that are the most useful. Full list of all APIs can be found in documentation [3]. To authorize an API, click on the "Go to Authorize" and select the API from the list.

IoT core API can be used for device management (such as deleting devices, modifying the names of devices and getting properties of devices) and controlling devices (getting the status of a device or sending a command to a device). Device control through IoT core API is possible only for devices based on TuyaOS (Tuya operating system). To control devices based on TuyaLink solution (MQTT gateway), which is the method used throughout this paper, Device Northbound Service API must be used. This issue will be described in detail in chapter 9.

Authorization token management API is used to get access tokens, which are used in authorization process in each API request. More on this process will be described in chapter 9.

Device northbound service is an API used for controlling TuyaLink based devices. The northbound interface of a device is extracted into a set of *capabilities*. Mainly 2 types of API calls are used for communication with the device through this API: Query device capability to get status of the device from the cloud and Execute device capability to send commands to a device.

### ■ 8.3.4 Assets

Assets are used to group devices in the cloud project. Asset is a topological tree structure with multiple layers. Assets can be used to grant permissions to user accounts to add, delete and transfer devices under an asset. Assets

are useful for larger cloud projects with larger number of devices, for example an asset could group devices in one part of the building, and we could grant management permission to this group of devices to specific user accounts. It is not necessary to group devices in assets for smaller projects.

To create an asset, click on the "Add asset" button on the left side on the screen, as shown on [Fig. 8.3]. Devices can be added to the asset in the "Devices" tab with blue "Add device" button. Authorization for an user account to manage devices in the selected asset can be added in the "Authorized users" tab with blue "Add authorization" button.

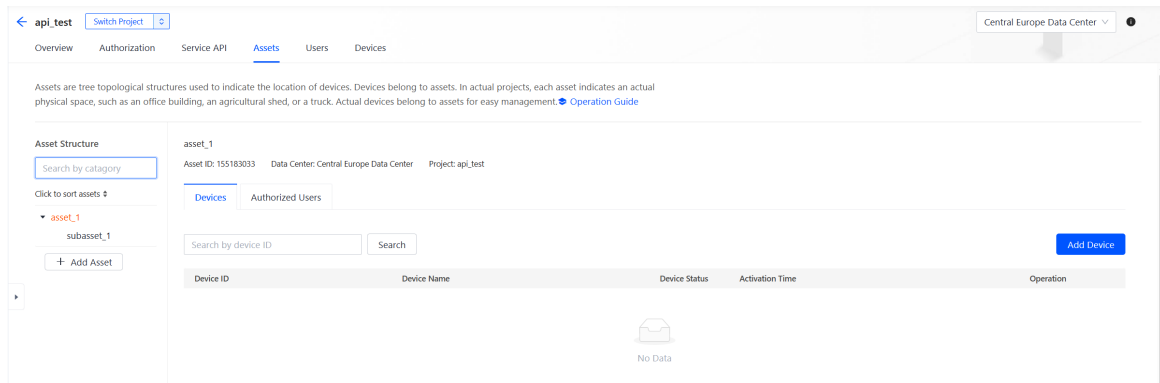


Figure 8.3: Asset setup menu

### 8.3.5 Devices

Devices can be added to the cloud project in the "Devices" tab. It is necessary to link devices to cloud project to be able to access their control interface through authorized APIs.

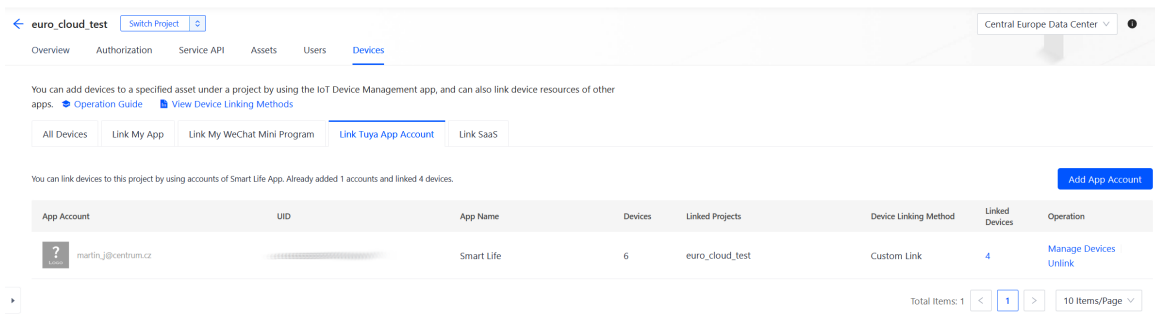
In the "All devices" tab is displayed a list of all devices currently linked to the this cloud project. New devices can be added by clicking on the blue "Add device" button. This way we can add virtual devices, which can be used for testing API calls. Physical devices can be added via Smart Industry App mobile application, however, I wasn't able to find this application in any app store, maybe it is not available in my region.

The other way to add physical devices to a cloud project, which worked out for me, is through Tuya Smart Life mobile application. To add devices to the cloud project, add them first to Tuya Smart Life app via QR code, as is described in chapter 11. Then go back to the "Devices" tab of the cloud development menu and select "Link Tuya App account" tab. To link a new Tuya app account, click on the blue "Add app account" button on the right [Fig. 8.4]. QR code will appear, scan it with a Tuya Smart Life app to link the app account to the current cloud project.

To link devices to the current cloud project from the linked app account, click on the "Manage devices" text on the right. New menu titled "Manage devices" will appear, as shown on [Fig. 8.5]. All devices currently linked to the Smart Life app will be displayed there. To link devices, check the

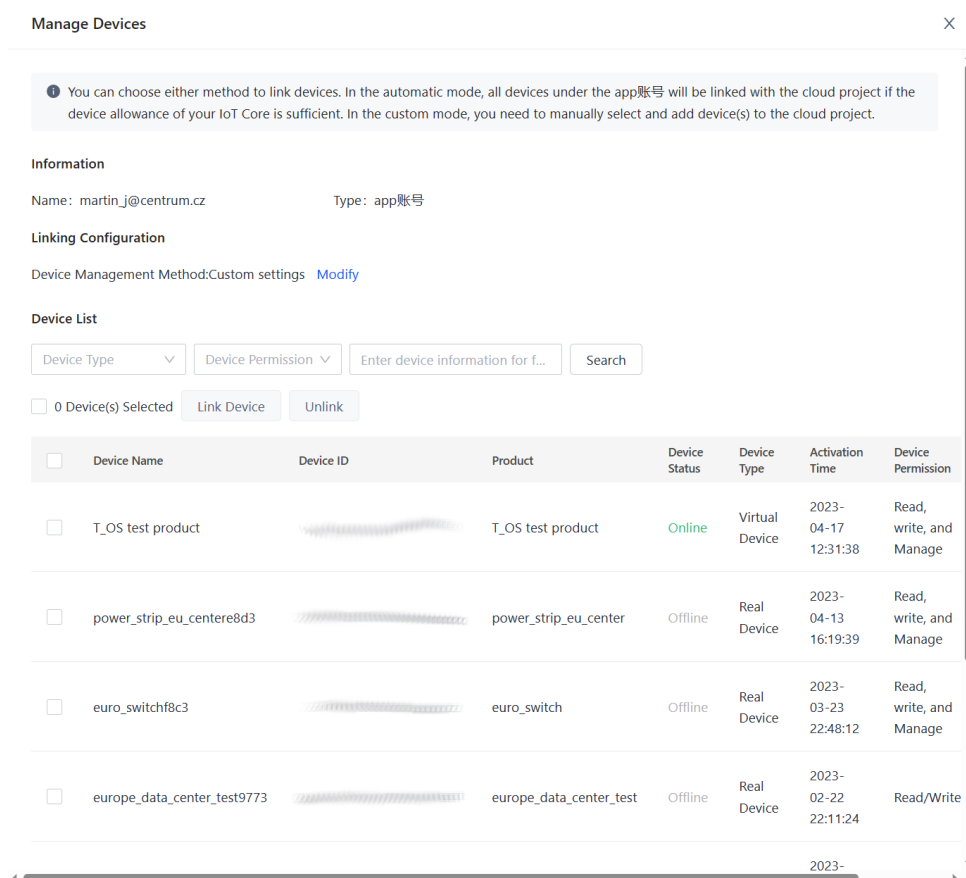


### 8.3. Sections of the Cloud development menu



**Figure 8.4:** Linking Tuya Smart Life app account to the cloud project

checkbox next to the name of the device and click on the "Link device" button, to unlink the device, click on the "Unlink" button.



**Manage Devices** X

① You can choose either method to link devices. In the automatic mode, all devices under the app账号 will be linked with the cloud project if the device allowance of your IoT Core is sufficient. In the custom mode, you need to manually select and add device(s) to the cloud project.

**Information**  
 Name: martin\_j@centrum.cz Type: app账号

**Linking Configuration**  
 Device Management Method: Custom settings [Modify](#)

**Device List**

Device Type Device Permission Enter device information for f... Search

0 Device(s) Selected [Link Device](#) [Unlink](#)

<input type="checkbox"/>	Device Name	Device ID	Product	Device Status	Device Type	Activation Time	Device Permission
<input type="checkbox"/>	T_OS test product		T_OS test product	Online	Virtual Device	2023-04-17 12:31:38	Read, write, and Manage
<input type="checkbox"/>	power_strip_eu_centere8d3		power_strip_eu_center	Offline	Real Device	2023-04-13 16:19:39	Read, write, and Manage
<input type="checkbox"/>	euro_switchf8c3		euro_switch	Offline	Real Device	2023-03-23 22:48:12	Read, write, and Manage
<input type="checkbox"/>	europe_data_center_test9773		europe_data_center_test	Offline	Real Device	2023-02-22 22:11:24	Read/Write

**Figure 8.5:** Linking devices from Tuya Smart Life account

## 8.4 API explorer

Tuya provides very useful tool for debugging API requests, which is called API explorer. API explorer can be accessed from the Cloud development menu. API explorer is used for making API requests from the web browser. To make an API call, select the API endpoint on the left [Fig. 8.6]. Fill out the parameters of the request and submit the request with the "Submit request" button. Request URL and response to the request can be viewed on the right. Documentation for each API endpoint can be viewed in the "View Docs" tab.

The screenshot displays the Tuya IoT Platform API Explorer interface. The top navigation bar includes the Tuya logo, 'IoT Platform', 'API Explorer', and the current project 'euro\_cloud\_test'. The left sidebar shows a navigation menu with 'Device Northbound Service' selected, and sub-items for 'Device Capability Management' and 'Device Capability Access'. The main content area is titled 'Query Device Capability List' and shows a 'Parameter (Request Method: GET)' section with a 'Params' table containing 'device\_id' and 'tags'. The 'Response' section displays a JSON object with the following structure:

```

{
  "result": {
    "capabilities": [
      {
        "capability_code": "sw1",
        "methods": [
          "get",
          "post",
          "event"
        ],
        "name": "sw1",
        "request": {
          "description": "",
          "name": "sw1",
          "type": "boolean"
        },
        "response": {
          "description": "",
          "name": "sw1",
          "type": "boolean"
        },
        "tags": [
          "original"
        ]
      }
    ]
  }
}

```

The interface also includes a 'Submit Request' button and a 'Debugging Result' section with a warning message about authorization keys.

Figure 8.6: API explorer



## Chapter 9

# API request structure and authentication

### 9.1 Introduction

There are a few ways to control devices connected to Tuya cloud. Tuya's own Smart Life mobile application can be used or the API of the Tuya cloud can be used. API Requests are HTTP requests sent to an Application Programming Interface (API) of the Tuya cloud in order to retrieve the state of a device or control a device connected to the Tuya cloud. In this chapter, I will describe how to form an API request to the Tuya cloud, how to get access token and how to create signature for the API requests.

### 9.2 Request structure

There are 4 HTTP request methods supported by the Tuya API: GET, PUT, POST and DELETE, although GET and POST method are used in most API requests to Tuya cloud. GET method is used to request data from the cloud, the query strings with additional information about the request are stored in the URL of the request. POST request is used to send data to the cloud, these data are stored in the body of the request.

Each API request consists of a target URL of API endpoint, a request header and POST requests also have a message body. Request header is used to pass additional information to the server. Informations in header are stored as key:value pairs.

### 9.3 Authentication method

Each API request must be signed to verify its authenticity and data integrity. Signature is a string that is placed in the header of the API request and is created from a hash of concatenated string, which is made of ClientID, access token, current timestamp, request method (e.g. GET, POST), SHA-256 hash of a message body and the target URL of the API endpoint. The method used for creating the signature is HMAC-SHA256, the key used to hash the concatenated string is the Client secret string, which can be found in Cloud development menu, as described in 8.3.2.

ClientID is obtained from cloud development menu, as is described in 8.3.2 and is specific to each cloud project. Access token is used to authorize each API request, it is a string granted by the Tuya cloud with an API call to the Token management API with the correct credentials used (ClientID and Client secret). Access token expires after 2 hours, after this time, it must be renewed with a new API request.

# Chapter 10

## Web control interface

### 10.1 Introduction

In this section, I will describe the implementation of web interface used for controlling the APC telnet power strip through Tuya cloud. The web interface is based on the Express backend web application framework, which is one of the most popular backend framework for Node.js. Node.js is a backend runtime environment, which executes JavaScript code outside of a web browser.

### 10.2 Project structure

In this section, I will describe the structure of files in the project directory of the web interface project.

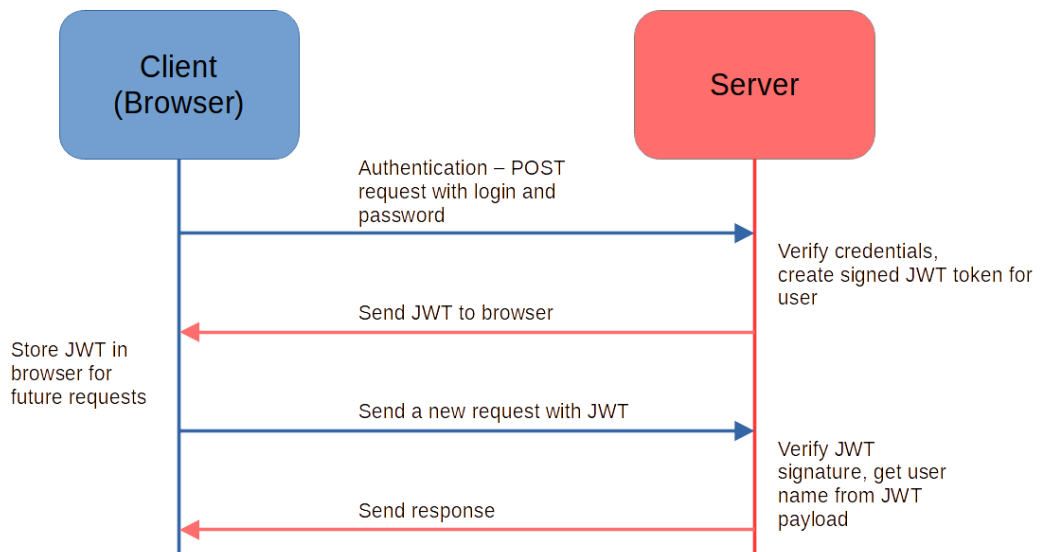
`server.js` is the main file, in which the Express server is initialized and run. The Views directory contains views, which are rendered by ejs template engine. Routes directory contain code, that is executed in reaction to the front end code making request to an endpoint with specific URL. There are 2 files in the routes directory: `tuya_api.js`, which is used for forming API requests to Tuya cloud, and `login.js`, which handles user login. Public directory contains static public files, such as `index.html` file for the login page, styles directory with `.css` style sheets and scripts directory with front end JavaScript scripts. Middleware directory contains middleware functions, such as `cookieJwtAuth.js` file, which is middleware function for verifying the validity of JWT tokens, which are used to ensure that only logged in users can access the website.

### 10.3 Authentication mechanism

To ensure that no unauthorized user could access the website and control devices through the website, authentication mechanism is implemented. Authentication process is shown on [Fig. 10.1].

Firstly, user must log in at the login page with valid credentials (username and password). Login page is made from a simple HTML form. Server verifies,

that the credentials are valid and creates a signed JSON web token (JWT), which is sent to the client browser. JWT has 3 parts: header, payload and signature. Header is in JSON format and identifies the type of algorithm used to generate the signature (usually HMAC or RSA signature). Payload is also in JSON format and can contain various information, such as username and expiration time. The signature is created using the cryptographic method specified in the header using the secret key stored on the server. Client browser stores the JWT locally and includes it with every request to the server. After receiving a request, the server verifies the JWT signature and pulls the username and other information from the payload of the JWT.



**Figure 10.1:** Authentication process with JWT tokens

Verification of JWT tokens is implemented in middleware function, which is shown in 10.1. Middleware is a piece of software, that runs between the time server receives the request and the time it sends the response. The function loads the JWT token from the cookie file, verifies the validity of the received JWT token on line 5. If the verification fails (typically when JWT token expires), cookie file, where the token is stored, is cleared and user is redirected to the login page.

```

1 const jwt = require("jsonwebtoken");
2 exports.cookieJwtAuth = (req, res, next) => {
3   const token = req.cookies.token;
4   try {
5     const user = jwt.verify(token, "secret_key");
6     req.user = user;
7     next();
8   } catch (err) {
9     res.clearCookie("token");
10    return res.redirect("/");}};

```

**Listing 10.1:** JWT middleware function



After inputting the correct credentials at the login page, JWT token is generated using the secret key stored at the server, expiration time of the token is set to 10 minutes, the token is saved in the cookie file in client's browser and user is redirected to the homepage, as shown in 10.2.

```
1 const token = jwt.sign(user, "secret_key", {expiresIn: "10m"});
2 res.cookie("token", token);
3 return res.redirect("/home");
```

**Listing 10.2:** Setting up the JWT token at the login page

## 10.4 Tuya API calls implementation

In this chapter I will describe how I implemented API calls to the Tuya cloud from the web control interface. I will also mention some issues I encountered along the way and how to solve them.

### 10.4.1 Cross-Origin Resource Sharing (CORS)

API requests to the Tuya cloud must be done from the backend part of the web application. There are mainly 2 reasons for this: sharing an access key with front end part of the application is generally not considered a safe practice and CORS policy of the Tuya cloud blocks this.

Cross-Origin Resource Sharing (CORS) is a mechanism, that allows web browser scripts to access resources from different domain than the one the application is running on [24]. By default, web browsers use same-origin policy for security reasons, which means they don't allow web browser scripts to access resources from a different domain. When attempting cross-origin request, extra *Origin* HTTP header is added with domain name of the source website. The server sends Access-Control-Allow-Origin (ACAO) header in the response, which contains either list of permitted origin domains, wildcard symbol indicating, that all domains are permitted for cross-origin request, or error message if the server doesn't allow cross-origin request.

It is not possible to call Tuya's API from the front end part of the web application, because the CORS policy set at the Tuya's cloud allows cross-origin requests only from IoT developer platform (<https://iot.tuya.com/>), which allows the use of debugging tool (API explorer) during the development.

### 10.4.2 Tuya API calls implementation

#### Introduction

The implementation of the API request will be described on the example of the *Execute Device Capability* API request, which is used to send a command to the device. This request uses POST method, which means that the request has a message body, as opposed to GET request, which is commonly used to get the access token and get the status of a device. API call is implemented into asynchronous function called *execute\_device\_capability*.

### ■ API selection issue

There are many API endpoints available on the Tuya cloud and sometimes it can be very confusing to select the correct one for the project. There are 2 APIs used for getting status and controlling devices: *IoT Core API* and *Device Northbound Service API*. I was trying to use the IoT Core API for my project, but it didn't work as expected. When getting the status of the device, the cloud returns response successfully, but the result field where the status of the device was always empty, as shown on listing 10.3. When trying to send command to a device, it would throw "Command or value not support" error. After communication with support of Tuya, I found out that IoT core API works only with TuyaOS based projects, which use TuyaOS operating system and proprietary network modules to handle communication.

```

1 {
2   "result": [],
3   "success": true,
4   "t": 1684328800785,
5   "tid": "aa0832eff4b311edb8e6f637b158923e"
6 }

```

**Listing 10.3:** Incorrect response from IoT Core API

When using TuyaLink based project (using MQTT gateway), Device Northbound Service API must be used to get the correct behavior. Correct response is shown on listing 10.4, it can be seen that cloud returns the status of the device as opposed to the response from IoT Core API, which returned empty result field. This information is hidden in different part of the documentation [1] than the main Cloud development documentation, so it can be quite confusing to select the correct API endpoint for the project.

```

1 {
2   "result": [
3     {
4       "value": false
5     }
6   ],
7   "success": true,
8   "t": 1684330553283,
9   "tid": "be99f6bef4b711edb8e6f637b158923e"
10 }

```

**Listing 10.4:** Correct response from Device Northbound Service API

### ■ Input parameters

This function takes 3 input parameters: access token, capability and value. Access token is obtained by separate API call to Token management API, which is implemented in the getToken function [6]. Capability is extracted from the model of a device, which is described in chapter 4 and specifies the id of the command that is sent to the device (e.g. change a property value, or perform an action). Value is the new value of the capability that is passed to the cloud in the message body. To control the power strip, id of the outlet

on the power strip is passed in the capability variable (sw1, sw2, ...) and True/False value is passed in the value parameter to turn on/off the power outlet.

### ■ Creating a signature

Message body is constructed and turned into JSON string on line 4. It is important to construct the message body properly, as failing to do so will result in confusing "Unknown error" in the response.

URL of the API endpoint is specified on the line 2, id of the device and capability name must be inserted into the URL path. Content of the message body is hashed using SHA-256 algorithm on line 5.

On the line 6 is created a new string named `stringToSign`, which is created by concatenation of HTTP request method, hashed content of the message body and URL of the API endpoint, as instructed in the documentation [21]. New string called `signatureBase`, which will be used for generating the signature is created on the line 11. `signatureBase` is created by concatenation of `ClientID`, access token, current 13-digit timestamp, nonce and `stringToSign` string. Nonce is a unique number, that can be used only once during secured communication, it improves the security, but for API requests to Tuya is optional and can be left blank.

Signature is created on the line 13 using the HMAC-SHA256 cryptographic method with previously mentioned string as the message body and `Client secret` as the key. How to find out `ClientID` and `client secret` is described in chapter 8.3.2. The hash is encoded in Base64 and capitalized, creating the final signature.

### ■ Making a request

The request is formed on line 16 using the `fetch` function. `Fetch` is one of the most used functions in JavaScript for HTTP requests. It is necessary to include the `ClientID`, signature, signature method (HMAC-SHA256), current timestamp, access token in the header of the request. For requests using `POST` method, it is also necessary to set `content-type` in the header to `application/json` to indicate the format of the message body. For `POST` type request, it is also necessary to send the message body in the request.

### ■ Handling expired access token

The response is received on line 31, if the error code of the response is 1010, the token is expired, `getToken()` function is called to request a new token and the function for making the API request is run again.

```

1 async function execute_device_capability(access_token,
    capability, value) {
2     const endpoint = new URL('https://openapi.tuya.com/v1.0/
    iot-03/devices/${config.deviceID}/capabilities/${capability
    }');
3

```

```

4   let body = JSON.stringify({"value":value});
5   let content_SHA256 = createHash('sha256').update(body).
    digest('hex');
6   const stringToSign = `POST\n${content_SHA256}\n\n/v1.0/iot
    -03/devices/${config.deviceID}/capabilities/${capability}`;
7
8   const nonce = "";
9   const t = Date.now();
10
11  const str = `${config.client_id}${access_token}${t}${nonce}${
    stringToSign}`;
12
13  var hash = CryptoJS.HmacSHA256(str, config.client_secret);
14  var hashInBase64 = hash.toString().toUpperCase();
15
16  const response = await fetch(endpoint, {
17    method: 'POST',
18    mode: 'cors',
19    headers: {
20      "client_id": config.client_id,
21      "sign": hashInBase64,
22      "sign_method": "HMAC-SHA256",
23      "t": t,
24      'content-type': 'application/json',
25      "access_token": access_token
26    },
27    body: body
28  });
29
30  const data = await response.json();
31  if (data.code === 1010) {
32    access_token = await getToken()
33    await execute_device_capability(access_token)
34  }
35 }

```

Listing 10.5: Implementation of API call

## ■ GET requests

Requests using GET method (used for getting access token, or state of the device) are formed very similarly, but there are a few key differences. Since GET requests don't have message body, the body is left as an empty string, which means that the SHA256 hash generated from it is always the same.

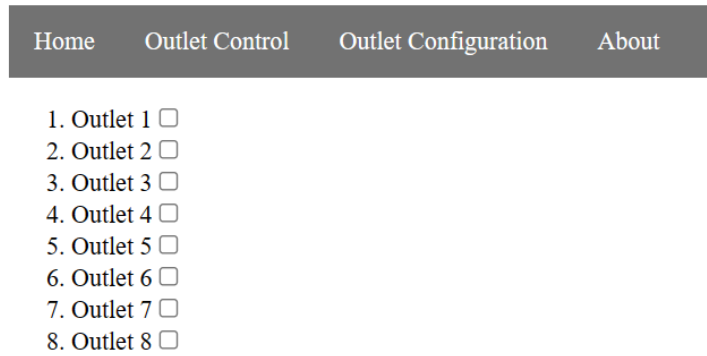
## ■ Access token requests

Requests for a new access token are generated the same way, but since a new access token is being requested, access token field is omitted from the request header and is omitted from the string used for creating a signature.

## 10.5 User interface

User interface consists of 8 simple toggle switches to turn on/off the power outlets, as shown on [Fig. 10.2].

### APC power outlet control panel



**Figure 10.2:** User interface of the web application



# Chapter 11

## Linking device to Tuya app

### 11.1 Introduction

In this chapter, I will discuss a few possible ways to control devices connected to the Tuya cloud via smartphone applications.

The simplest way is to link a device with Tuya's official application, which is called Tuya Smart Life. This application is available both for Android and iOS systems. Devices are linked by scanning a QR code, which is unique to each device.

For developers, who want to create a custom application, Tuya provides IoT App SDK (software development kit) to aid in the development of the application [9]. SDK is a collection of software development tools in one installable package [25].

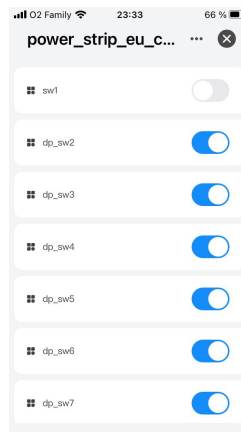
Other way to implement an application for controlling Tuya devices would be to implement a new application from the ground up and use API calls to control and monitor devices.

### 11.2 Linking devices to Tuya Smart Life application

Tuya Smart Life application is the default mobile application used for interaction with smart devices connected to Tuya cloud.

User interface for each device is generated from the set of functions assigned to the device (function assignment described in chapter 6.2.1). Custom user interface panels are unfortunately locked behind a subscription plan, but the default interface works just fine, although it looks quite plain.

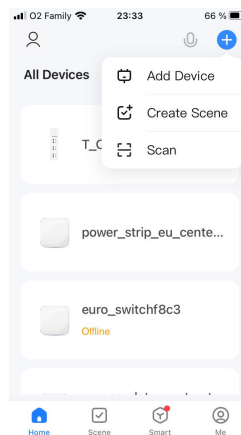
The user interface of the consists of 8 toggle switches used to turn on/off each of the power outlet [Fig. 11.1]



**Figure 11.1:** Control interface of the APC power strip in Tuya Smart Life app

To link a device with Tuya Smart Life application, go to the Device management tab of the Product development menu, as is described in chapter 6.2.3. Choose a device from the list of devices and click on the blue "QR code binding text" on the right side of the screen, next to the selected device to be linked, the QR code will show up.

To add the device to the Tuya Smart Life application, scan the QR code with the Smart Life application on the phone, as is shown on [Fig. 11.2].



**Figure 11.2:** Linking a device to the Tuya Smart Life application



# Chapter 12

## Conclusion

The aim of this thesis was to propose and implement a solution for integration of devices with non-standard interfaces into the Tuya IoT cloud platform.

The first part of the assignment was to provide detailed description of the implementation of a generic device connected to Tuya cloud platform with instructions both on the device side and on the Tuya cloud side.

MQTT client for Tuya IoT cloud platform was implemented using the Paho MQTT library into Python module. This module can be very easily incorporated into any Python code, making for very easy implementation of Tuya cloud connectivity into any device, that can run Python code and supports TCP/IP stack. Detailed description of the implementation of a generic device connected to the Tuya cloud platform was provided in chapter 5.

Detailed instructions on how to set up a new device on the Tuya cloud platform were provided in chapter 6.

The second part of the assignment was to implement IoT Tuya gateway for remote control of APC 230V power outlets with Telnet interface. IoT gateway was implemented on the Raspberry Pi hardware, which was used to connect the power outlets to the Tuya cloud platform. The implementation was described in chapter 7.

Third part of the assignment was to implement a web interface to allow remote switching of the APC 230V power outlets. Control interface for remote switching of APC 230V power outlets with Telnet interface was implemented into a web application in chapter 10.

The fourth part of the assignment was to use smartphone application to allow switching of the above outlets. For remote switching of power outlets from a smartphone application, Tuya Smart Life application was used. How to link a device defined on the Tuya cloud platform with the Tuya Smart Life application is described in chapter 11.

All goals set in the assignment were accomplished.





## Bibliography

- [1] *Application Development (TuyaLink)*. URL: <https://developer.tuya.com/en/docs/iot/application-dev?id=Kbf53a58zz6t1> (visited on 05/17/2023).
- [2] Wesley Chai, Kate Brush, and Stephen J. Bigelow. *What is PaaS? Platform as a service definition and guide*. URL: <https://www.techtarget.com/searchcloudcomputing/definition/Platform-as-a-Service-PaaS> (visited on 12/09/2022).
- [3] *Cloud Services API Reference*. URL: <https://developer.tuya.com/en/docs/cloud> (visited on 05/03/2023).
- [4] *Data parsing*. URL: <https://developer.tuya.com/en/docs/iot/Data-Parsing?id=Kb4qgsj9g1duj%5C#title-1-Data%5C%20parsing> (visited on 02/15/2023).
- [5] *Directly-connected device authentication*. URL: <https://developer.tuya.com/en/docs/iot/MQTT-protocol?id=Kb65nphxrj8f1%5C#title-10-Directly-connected%5C%20device%5C%20authentication> (visited on 12/09/2022).
- [6] *Get a Token*. URL: <https://developer.tuya.com/en/docs/cloud/6c1636a9bd?id=Ka7kjumkoa53v> (visited on 05/16/2023).
- [7] *Glossary - Cloud development*. URL: <https://developer.tuya.com/en/docs/iot/terms?id=K914joq6tegj4> (visited on 02/14/2023).
- [8] *Introducing JSON*. URL: <https://www.json.org/json-en.html>.
- [9] *IoT App SDK*. URL: <https://developer.tuya.com/en/docs/iot/app-sdk-instruction?id=K9kjstc7t376p> (visited on 05/21/2023).
- [10] Martin Malý. “Protokol MQTT: komunikační standard pro IoT”. In: (). URL: <https://www.root.cz/clanky/protokol-mqtt-komunikacni-standard-pro-iot/> (visited on 12/06/2022).
- [11] *MQTT Version 3.1.1 OASIS Standard*. URL: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html> (visited on 12/07/2022).

- [12] *MQTT Version 3.1.1 OASIS Standard - 2.1 Structure of an MQTT Control Packet*. URL: [http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#\\_Toc398718019](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718019) (visited on 12/07/2022).
- [13] *MQTT Version 3.1.1 OASIS Standard - 3.1 CONNECT – Client requests a connection to a Server*. URL: [http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#\\_Toc398718028](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718028) (visited on 12/07/2022).
- [14] *MQTT Version 3.1.1 OASIS Standard - 3.2 CONNACK – Acknowledge connection request*. URL: [http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#\\_Toc398718033](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718033) (visited on 12/07/2022).
- [15] *MQTT Version 3.1.1 OASIS Standard - 3.3 PUBLISH – Publish message*. URL: [http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#\\_Toc398718037](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718037) (visited on 12/07/2022).
- [16] *MQTT Version 3.1.1 OASIS Standard - 4.3 Quality of Service levels and protocol flows*. URL: [http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#\\_Toc398718099](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718099) (visited on 12/07/2022).
- [17] *Paho Python MQTT Client-Understanding The Loop*. URL: <http://www.steves-internet-guide.com/loop-python-mqtt-client/> (visited on 05/19/2023).
- [18] *Remote configuration*. URL: [https://developer.tuya.com/en/docs/iot/remote\\_config?id=Kbrxvzug63axu](https://developer.tuya.com/en/docs/iot/remote_config?id=Kbrxvzug63axu) (visited on 02/14/2023).
- [19] *Scene Linkage*. URL: <https://developer.tuya.com/en/docs/iot/Scenario-connection-settings?id=Kbr989qepvih9> (visited on 02/14/2023).
- [20] *Secure connection*. URL: <https://developer.tuya.com/en/docs/iot/MQTT-protocol?id=Kb65nphxrj8f1%5C#title-13-Secure%5C%20connection> (visited on 12/09/2022).
- [21] *Sign Requests*. URL: <https://developer.tuya.com/en/docs/iot/new-signature?id=Kbw0q34cs2e5g> (visited on 05/16/2023).
- [22] *TuyaOS development*. URL: <https://developer.tuya.com/en/docs/iot/embedded-software-development?id=Ka5nw43r01smp> (visited on 12/09/2022).
- [23] *What is Tuya IoT Edge Gateway?* URL: <https://developer.tuya.com/en/docs/iot/overview?id=Kag6f93nyhhcq> (visited on 02/02/2023).
- [24] Wikipedia contributors. *Cross-origin resource sharing — Wikipedia, The Free Encyclopedia*. [Online; accessed 16-May-2023]. 2023. URL: [https://en.wikipedia.org/w/index.php?title=Cross-origin\\_resource\\_sharing&oldid=1138566357](https://en.wikipedia.org/w/index.php?title=Cross-origin_resource_sharing&oldid=1138566357).

- [25] Wikipedia contributors. *Software development kit* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 21-May-2023]. 2023. URL: [https://en.wikipedia.org/w/index.php?title=Software\\_development\\_kit&oldid=1148272226](https://en.wikipedia.org/w/index.php?title=Software_development_kit&oldid=1148272226).
- [26] Wikipedia contributors. *Telnet* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 17-May-2023]. 2023. URL: <https://en.wikipedia.org/w/index.php?title=Telnet&oldid=1154644574>.