Labkit

User Guides

Author: Labkit Team

Date: Mar 2023

Version: 1.0

1. Context

To understand the user journeys and the APIs called in LabKit, below are their descriptions and usage explanations.

2. Devices Journey

2.1 Device Registration and Verification

2.1.1 By SDK

The registration and verification of a device can be performed via API (2.1.2) or via SDK. Our recommendation is that it be executed via SDK as in the example below, as it does not require hardware integration.

```
./dab-app-wrapper -r
```

r- registration

2.1.2 By API

This operation is used whenever a device wishes to register itself in DAB in order to use the DAB platform, i.e. it allows dab to create the device's identity on the platform.

API: POST /devices

2.1.2.1 Request body

```
  "codeType": "IMSI",
  "code": "901280063661846",
  "simKey": "-----BEGIN PUBLIC KEY-----MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEktAwQ73e2D
DhD5I4G/Q9oTGYOrMn7CyUQoHYZ8x8lKmAWJt9q3vohG+hp/L+OeqW7nva9jwmKyIcPf6XgabO4A==-----EN
D PUBLIC KEY-----",
  "verificationMethod": "SMS",
  "signingMethod": "SIM_TRUST",
  "info": {}
```

2.1.2.2 Responses

Regarding other error handling, all codes have the same return object pattern: code, type, message and details (optional)

Code (http)	Description
201	Created `{ "deviceId": "0af401df-7d7a-1f36-817d-7b0a058d0003", "ncrypt": "0af401df-7d7a-1f36-817d-7b0a058d0003" }`
400	Parameters missing in request
401	Unauthorized
403	Forbidden
409	There is already a device registered with same code and code type in DAB
500	Unexpected errors not mapped to status codes mentioned above

It is possible to verify two practical examples: programmatic via script or visual through the graphical interface (feature in the menu - Register Device)

Script: device_registration.py

2.2 Check Device's status/info

This operation is used whenever the Platform wants to retrieve a list of devices owned by an Organisation, in order to view device's information.

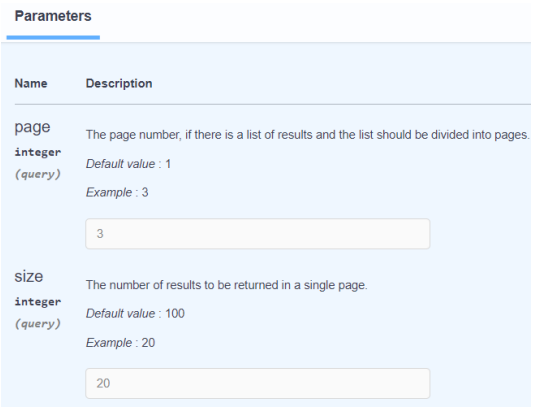
Organization is the same as account, although they are different entities in practice they are the same thing: dab account.

The API POST / devices/query used whenever the Platform wants to retrieve a filtered list of devices, and the API GET /devices/{deviceID} is used whenever the Platform wants to retrieve the details of a specific device.

2.2.1.1 Parameters

API: GET /devices

In this case, there are paging parameters: page and size.



2.2.1.2 Responses

Regarding other error handling, all codes have the same return object pattern: code, type, message and details (optional)

Code (http)	Description

200	OK
	{
	"content": [
	{
	"id": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"code": "901280063661846",
	"codeType": "IMSI",
	"simKey": "-----BEGIN PUBLIC KEY-----MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEktAwQ73e2DDhD5I4G/Q9oTG YOrMn7CyUQoHYZ8x8lKmAWJt9q3vohG+hp/L+OeqW7nva9jwmKyIcPf6XgabO4A=======END PUBLIC KEY-----",
	"verificationMethod": "SMS",
	"signingMethod": "SIM_TRUST",
	"status": "UNASSIGNED",
	"dabAccountId": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"deviceWallet": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"dabAccountWallet": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"alternativeIds": [
	"dab:device:name:myCar"
	],
	"ncrypt": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"balance": [
	{
	"amount": 1000,
	"tokenId": "VDF_TOKEN",
	"issuer": "OU=DAB Sandbox Node 1, O=Fifth-9 Limited UK, L=London, ST=London, C=GB",
	"fractionDigits": 0
	}
	],
	"userId": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"grants": [
	{
	"grantId": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"serviceId": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"paymentType": "VISA_TOKEN",
	"status": "ACTIVE",
	"expirationDate": "2031-12-10T11:04:33.842",
	"roles": [
	{
	"name": "ADMIN",
	"permissions": [
	"REQUEST_PAYMENT"
	]
	}
	]
	}
	],
	"expirationDate": "2031-12-10T11:04:33.842",
	"registrationDate": "2031-12-10T11:04:33.842",
	"info": {}
	}
	],
	"totalResults": 50,
	"pageNumber": 3,
	"pageSize": 20
	}

401	Unauthorized
403	Forbidden
500	Unexpected errors not mapped to status codes mentioned above

2.2.1.3 Parameters

API: POST /devices/query

2.2.1.4 Request body


```
{
  "matchAlternativeId": "dab:device:name:myCar",
  "matchDeviceIds": [
    "0af401df-7d7a-1f36-817d-7b0a058d0003"
  ],
  "matchServiceId": "0af401df-7d7a-1f36-817d-7b0a058d0003",
  "matchDeviceStatus": [
    "UNASSIGNED"
  ],
  "matchXDeviceCode": "IMSI:123456789",
  "pageSize": 20,
  "pageNumber": 3
}
```


2.2.1.5 Responses

Regarding other error handling, all codes have the same return object pattern: code, type, message and details (optional)

Code (http)	Description

200	{
	"content": [
	{
	"id": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"code": "901280063661846",
	"codeType": "IMSI",
	"simKey": "-----BEGIN PUBLIC KEY-----MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEktAwQ73e2DDhD5I4G/Q9oTGYOrMn7CyUQoHYZ8x8lKmAWJt9q3vohG+hp/L+OeqW7nva9jwmKyIcPf6XgabO4A==------END PUBLIC KEY-----",
	"verificationMethod": "SMS",
	"signingMethod": "SIM_TRUST",
	"status": "UNASSIGNED",
	"dabAccountId": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"deviceWallet": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"dabAccountWallet": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"alternativeIds": [
	"dab:device:name:myCar"
	],
	"ncrypt": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"balance": [
	{
	"amount": 1000,
	"tokenId": "VDF_TOKEN",
	"issuer": "OU=DAB Sandbox Node 1, O=Fifth-9 Limited UK, L=London, ST=London, C=GB",
	"fractionDigits": 0
	}
	],
	"userId": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"grants": [
	{
	"grantId": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"serviceId": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"paymentType": "VISA_TOKEN",
	"status": "ACTIVE",
	"expirationDate": "2031-12-10T11:04:33.842",
	"roles": [
	{
	"name": "ADMIN",
	"permissions": [
	"REQUEST_PAYMENT"
	]
	}
	]
	}
	],

	"expirationDate": "2031-12-10T11:04:33.842", "registrationDate": "2031-12-10T11:04:33.842", "info": {} }], "totalResults": 50, "pageNumber": 3, "pageSize": 20 }
401	Unauthorized
403	Forbidden
500	Unexpected errors not mapped to status codes mentioned above

2.2.1.6 Parameters

API: GET /devices/{deviceID}

deviceId (string) - unique identifier of the device in the Corda DLT and DAB database

Example : 0af401df-7d7a-1f36-817d-7b0a058d0003

2.2.1.7 Responses

Regarding other error handling, all codes have the same return object pattern: code, type, message and details (optional)

Code (http)	Description

200	{
	"id": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"code": "901280063661846",
	"codeType": "IMSI",
	"simKey": "-----BEGIN PUBLIC KEY-----MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEktAwQ73e2DDhD5I4G/Q9oTGYOrMn7CyUQoHYZ8x8lKmAWJt9q3vohG+hp/L+OeqW7nva9jwmKyIcPf6XgabO4A==------END PUBLIC KEY-----",
	"verificationMethod": "SMS",
	"signingMethod": "SIM_TRUST",
	"status": "UNASSIGNED",
	"dabAccountId": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"deviceWallet": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"dabAccountWallet": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"alternativeIds": [
	"dab:device:name:myCar"
	],
	"ncrypt": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"balance": [
	{
	"amount": 1000,
	"tokenId": "VDF_TOKEN",
	"issuer": "OU=DAB Sandbox Node 1, O=Fifth-9 Limited UK, L=London, ST=London, C=GB",
	"fractionDigits": 0
	}
	],
	"userId": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"grants": [
	{
	"grantId": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"serviceId": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"paymentType": "VISA_TOKEN",
	"status": "ACTIVE",
	"expirationDate": "2031-12-10T11:04:33.842",

```
      "roles": [
        {
          "name": "ADMIN",
          "permissions": [
            "REQUEST_PAYMENT"
          ]
        }
      ]
    }
  ],
  "expirationDate": "2031-12-10T11:04:33.842",
  "registrationDate": "2031-12-10T11:04:33.842",
  "info": {}
}
```


401	Unauthorized
403	Forbidden
404	Resource not found
500	Unexpected errors not mapped to status codes mentioned above

It is possible to verify two practical examples: programmatic via script or visual through the graphical interface (feature in the menu – Status)

Scripts: device_list_by_query.py

device_details.py

2.3 Triggering a service via Business Transaction

2.3.1 By SDK

The action of triggering a service through Business Transactions can be performed via API (2.3.1) or by our recommendation via SDK, as in the example below:

```
./dap-app-wrapper -a EvCharging -t destination UUID y894y29843 -t action STRING START_CHARGE
```

a- service name

t- triggers (each trigger is composed of name, type and value) and can be multiple

2.3.2 By API

This operation allows sending data in a reliable, verifiable and guaranteed way by the device to trigger a service.

2.3.2.1 Parameters

API: POST /transactions

X-Device-Code (string) - Format is Type:Value

Example : IMSI:123456789 OR
SIMKEY:865870559gjdfmc4c04ymb8vm5040c5fc5476t6436xh46v87n

2.3.2.2 Request body


```
{
  "action": "evCharging",
  "payload": "eyJhbGciOiJFUzI1NiIsInNpZ25pbmdNZXRob2QiOiJJT1RfU0FGRSIsInR5cCI6IkpXVCJ
9.eyJ0cmlnZ2VycyI6W3sibmFtZSI6ImNvbm5lY3RvcklkIiwidmFsdWUiOiIzIiwidHlwZSI6IlNUUklORyJ
9LHsibmFtZSI6ImRlc3RpbmF0aW9uIiwidmFsdWUiOiIwYWY0MDIxNi03ZmU1LTE1NDAtODE3Zi1lNThjYjQw
YzAwMDUiLCJ0eXBlIjoiVVVJRCJ9LHsibmFtZSI6Im9mZmNoYWluVHJpZ2dlciIsInZhbHVlIjoidHJ1ZSIsI
nR5cGUiOiJCT09MRUFOIn1dLCJkZWNpc2lvbktleSI6ImV2Q2hhcmdlIn0.U9dVyl_qbX7pQD7wghhVWK1vA2
6G1UGsJWLD1sQR8J5HKE2wgs7Ug59DnkS0rbCGwegdoogYkMn37CQbfJClhQ"
}
```


2.3.2.3 Responses

Regarding other error handling, all codes have the same return object pattern: code, type, message and details (optional)

Code (http)	Description
204	No Content Headers: Name: X-Dab-Ncrypt — Description: new ncrypt generated by DLT — Type: string
400	Missing mandatory paramenters
401	Unauthorized
403	Forbidden
404	Not found

It is possible to verify two practical examples: programmatic via script or visual through the graphical interface (feature in the menu – Business Transaction).

Script: device_trigger_transaction

3. Platform Journey

3.1 Device Management

3.1.1 Assign/Reassign Device to a Dab Account

This operation is used whenever the Platform wants to associate a DAB account to a device. At this point, the device is registered in the DLT. Device cannot be in DEACTIVATED status.

3.1.1.1 Parameters

API: POST /devices/assignment

3.1.1.2 Request body

```
[
  "0af401df-7d7a-1f36-817d-7b0a058d0003"
]
```

3.1.1.3 Responses

Regarding other error handling, all codes have the same return object pattern: code, type, message and details (optional)

Code (http)	Description
201	Created `{ "message": "Device <> Account association. 0 of 1 were successful and took 4 seconds", "deviceList": [{ "deviceId": "0af401df-7d7a-1f36-817d-7b0a058d0003", "ncrypt": "0af401df-7d7a-1f36-817d-7b0a058d0003", "failureMessage": "Assignment of Device with id 0af40181-7db3-1ecc-817d-be5d44510002 and Account with id 5478635b-1f83-4cba-bbd2-716b8f48e182 failed. A device with IMSI code 901280063661844 was already registered! at 2022-01-13T18:08:28.388Z" }] }`
400	Parameters missing in request or pair code/code type doesn't belong to GDSP account
401	Unauthorized
403	Forbidden
404	Resource not found, either the device or the DAB account

409	There is already a device registered with same code and code type in DAB
500	Unexpected errors not mapped to status codes mentioned above

It is possible to verify two practical examples: programmatic via script or visual through the graphical interface (feature in the menu - Manage Device – Assign Device)

Script: device_account_association.py

3.1.2 Update Devices Info

This operation is used whenever the Platform wants to update some of the device's attributes or whenever the WoW app wants to update the device name.

3.1.2.1 Parameters

API: PATCH /devices/{deviceId}

deviceId (string) - unique identifier of the device in the Corda DLT and DAB database

Example : 0af401df-7d7a-1f36-817d-7b0a058d0003

3.1.2.2 Request body

```
{
  "code": "901280063661846",
  "codeType": "IMSI",
  "simKey": "-----BEGIN PUBLIC KEY-----MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEktAwQ73e2D
DhD5I4G/Q9oTGYOrMn7CyUQoHYZ8x8lKmAWJt9q3vohG+hp/L+OeqW7nva9jwmKyIcPf6XgabO4A==-----EN
D PUBLIC KEY-----",
  "verificationMethod": "SMS",
  "info": {},
  "expirationDate": "2031-12-10T11:04:33.842",
  "alternativeIds": [
    "dab:device:name:myCar"
  ]
}
```

3.1.2.3 Responses

Regarding error handling, all codes have the same pattern as the return object: code, type, message and details(optional)

Code (http)	Description
204	No Content
400	Parameters missing in request or request is malformed. For example, the alternativeId with the device name is not in the correct format or more attribute other than the device name are tried to be updated via WoW app
401	Unauthorized
403	Forbidden

404	Resource not found
409	There is already a device registered with same code and code type or sim key or update is not allowed due to device's status
500	Unexpected errors not mapped to status codes mentioned above

It is possible to verify two practical examples: programmatic via script or visual through the graphical interface (feature in the menu - Manage Device – Update Device)

Script: device_update.py

3.1.3 Deactivate/Reactivate Device

The API DELETE /devices/{deviceId}/activation is used whenever the Platform wants to deactivate a device. Device must be in UNASSIGNED, VERIFIED or EXPIRED status.

The API POST /devices/{deviceId}/activation is used whenever the Platform wants to reactivate a device. Device must be in DEACTIVATED status.

3.1.3.1 Parameters

URL: DELETE /devices/{deviceId}/activation

deviceId (string) - unique identifier of the device in the Corda DLT and DAB database

Example : 0af401df-7d7a-1f36-817d-7b0a058d0003

3.1.3.2 Responses

Regarding error handling, all codes have the same pattern as the return object: code, type, message and details(optional)

Code (http)	Description
204	No Content
401	Unauthorized
403	Forbidden
404	Resource not found
409	Deactivation not allowed due to device's status
500	Unexpected errors not mapped to status codes mentioned above

3.1.3.3 Parameters

URL: POST /devices/{deviceId}/activation

deviceId (string) - unique identifier of the device in the Corda DLT and DAB database

Example : 0af401df-7d7a-1f36-817d-7b0a058d0003

3.1.3.4 Responses

Regarding error handling, all codes have the same pattern as the return object: code, type, message and details(optional)

Code (http)	Description
204	No Content
401	Unauthorized
403	Forbidden
404	Resource not found
409	Deactivation not allowed due to device's status
500	Unexpected errors not mapped to status codes mentioned above

It is possible to verify two practical examples: programmatic via script or visual through the graphical interface (feature in the menu - Manage Device – Deactivate/Reactivate)

Scripts: device_activation.py

device_deactivation.py

3.1.4 List/Filter devices

In this operation, the same APIs as in the Check Device's status/Info case are called: GET /devices and POST /devices/query. These are described in point 2.2 above.

It is possible to verify two practical examples: programmatic via script or visual through the graphical interface (feature in the menu – Manage Device - List)

Script: device_list.py

3.2 Services (“Smart Contracts”) Lifecycle management

3.2.1 Create Service

This operation is used whenever the Platform wants to create an service, i.e. it is the business logic created by the partner that will be accessible by the device through business transactions and is similar to the smart contract concept.

API: POST /services

3.2.1.1 Request body

```
{
  "name": "Parking Service",
  "businessLogic": "consult https://confluence.tools.aws.vodafone.com/display/DP/DAB-+Data+Dictionary",
  "expirationDate": "2031-12-10T11:04:33.842",
  "acceptedPaymentMethods": [
    {
      "paymentType": "VISA_TOKEN"
    }
  ],
  "description": "string"
}
```

3.2.1.2 Responses

Regarding other error handling, all codes have the same return object pattern: code, type, message and details (optional)

Code (http)	Description
201	Created `{ "serviceId": "0af401df-7d7a-1f36-817d-7b0a058d0003" }`
400	Parameters missing in request
409	Service with the same decision key or name already exists
500	Unexpected errors not mapped to status codes mentioned above

It is possible to verify two practical examples: programmatic via script or visual through the graphical interface (feature in the menu - Manage Service – Create Service)

Script: service_creation.py

3.2.2 List/Filter Services

The API GET /services/{serviceId} is used whenever the Platform wants to retrieve the details of a specific service.

The API POST /services/list is used to list services according to specific filters.

3.2.2.1 Parameters

API: GET /services/{serviceId}

serviced (string) - unique identifier of the service

3.2.2.2 Responses

Regarding other error handling, all codes have the same return object pattern: code, type, message and details (optional)

Code (http)	Description

200	Ok
	<pre>{
	  "identifier": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	  "name": "Parking Service",
	  "status": "ACTIVE",
	  "triggers": [
	  {
	  "name": "chargedValue",
	  "type": "UUID",
	  "value": "50"
	  }
	  ],
	  "businessLogic": "consult https://confluence.tools.aws.vodafone.com/display/DP/DAB-+Data+Dictionary",
	  "expirationDate": "2031-12-10T11:04:33.842",
	  "acceptedPaymentMethods": [
	  {
	  "paymentType": "VISA_TOKEN"
	  }
	  ],
	  "serviceOwnerDabAccountId": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	  "registrationHost": "OU=DAB Sandbox Node 1, O=Fifth-9 Limited UK, L=London, ST=London, C=GB",
	  "description": "string"
	}</pre>

404	Resource not found
500	Unexpected errors not mapped to status codes mentioned above

3.2.2.3 Parameters

API: POST /services/list

3.2.2.4 Request body

<pre>{</pre>

```
  "matchServiceId": [
    "0af401df-7d7a-1f36-817d-7b0a058d0003"
  ],
  "matchServiceStatus": [
    "ACTIVE"
  ],
  "pageSize": 20,
  "pageNumber": 3
}
```


3.2.2.5 Responses

Regarding other error handling, all codes have the same return object pattern: code, type, message and details (optional)

Code (http)	Description

200	Ok
	{
	"content": [
	{
	"identifier": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"name": "Parking Service",
	"status": "ACTIVE",
	"triggers": [
	{
	"name": "chargedValue",
	"type": "UUID",
	"value": "50"
	}
	],
	"businessLogic": "consult https://confluence.tools.aws.vodafone.com/display/DP/DAB-+Data+Dictionary",
	"expirationDate": "2031-12-10T11:04:33.842",
	"acceptedPaymentMethods": [
	{
	"paymentType": "VISA_TOKEN"
	}
	],
	"serviceOwnerDabAccountId": "0af401df-7d7a-1f36-817d-7b0a058d0003",
	"registrationHost": "OU=DAB Sandbox Node 1, O=Fifth-9 Limited UK, L=London, ST=London, C=GB",
	"description": "string"
	}
	],
	"totalResults": 10,
	"pageNumber": 3,
	"pageSize": 20
	}

401	Unauthorized
403	Forbidden
500	Unexpected errors not mapped to status codes mentioned above

It is possible to verify two practical examples: programmatic via script or visual through the graphical interface (feature in the menu - Manage Service – List)

Scripts: service_list.py

service_details.py

3.2.3 Update Service

This operation is used whenever a third party /costumer service wants to update some of the service's attributes. Service has to be in INACTIVE status.

3.2.3.1 Parameters

API: PATCH /services/{serviceId}

serviced (string) - unique identifier of the service

3.2.3.2 Request body

```
{
  "name": "Parking Service",
  "businessLogic": "consult https://confluence.tools.aws.vodafone.com/display/DP/DAB-
+Data+Dictionary",
  "expirationDate": "2031-12-10T11:04:33.842",
  "acceptedPaymentMethods": [
    {
      "paymentType": "VISA_TOKEN"
    }
  ],
  "description": "string"
}
```

3.2.3.3 Responses

Regarding error handling, all codes have the same pattern as the return object: code, type, message and details(optional)

Code (http)	Description
204	No Content
400	Parameters missing in request
401	Unauthorized
403	Forbidden
404	Resource not found
409	There is already a device registered with same code and code type or sim key or update is not allowed due to device's status
500	Unexpected errors not mapped to status codes mentioned above

It is possible to verify two practical examples: programmatic via script or visual through the graphical interface (feature in the menu - Manage Service – Update Service)

Script: service_update.py

3.2.4 Subscribe device to a Service

This operation is used to Creats a subcription for a certain service.

3.2.4.1 Parameters

API: POST /devices/service-subscription

3.2.4.2 Request body

```
{
  "serviceId": "0af401df-7d7a-1f36-817d-7b0a058d0003",
  "deviceId": "0af401df-7d7a-1f36-817d-7b0a058d0003",
  "paymentType": "VISA_TOKEN"
}
```

3.2.4.3 Responses

Regarding other error handling, all codes have the same return object pattern: code, type, message and details (optional)

Code (http)	Description
201	Created `{ "name": "associateAccountToDevice", "messages": "Trace ID [173ccc30-b537-4b88-86c3-421e23be40f0] :: Operation [associateAccountToDevice] :: Assignment of Device with id 0af40181-7db3-1ecc-817d-be5d44510002 and Account with id 5478635b-1f83-4cba-bbd2-716b8f48e182 failed. A device with IMSI code 901280063661844 was already registered! at 2022-01-13T18:08:28.388Z", "totalMessages": 1, "resultMessage": "Device <> Account association. 0 of 1 were successful and took 4 seconds", "totalRequests": "1" }`
401	Unauthorized
403	Forbidden
500	Unexpected errors not mapped to status codes mentioned above

It is possible to verify two practical examples: programmatic via script or visual through the graphical interface (feature in the menu - Manage Service – Subscribe Service)

Scripts: device_service_subscription.py

3.2.5 Update (or remove) device subscription to a service

The API DELETE /devices/{deviceId}/grants/{grantId} is used whenever the Platform wants to delete the service <> device association.

The API PATCH /devices/{deviceId}/grants/{grantId} is used whenever the Platform/ costumer service wants to update the service <> device association.

3.2.5.1 Parameters

API: DELETE /devices/{deviceId}/grants/{grantId}

deviceId (string) - unique identifier of the device in the Corda DLT and DAB database - Example : 0af401df-7d7a-1f36-817d-7b0a058d0003

grantId (string) - unique identifier of the grant in the Corda DLT –

Example : 0af401df-7d7a-1f36-817d-7b0a058d0003

3.2.5.2 Responses

Regarding error handling, all codes have the same pattern as the return object: code, type, message and details(optional)

Code (http)	Description
204	No Content
401	Unauthorized
403	Forbidden
404	Resource not found
500	Unexpected errors not mapped to status codes mentioned above

3.2.5.3 Parameters

API: PATCH /devices/{deviceId}/grants/{grantId}

deviceId (string) - unique identifier of the device in the Corda DLT and DAB database - Example : 0af401df-7d7a-1f36-817d-7b0a058d0003

grantId (string) - unique identifier of the grant in the Corda DLT - Example : 0af401df-7d7a-1f36-817d-7b0a058d0003

3.2.5.4 Request body

```
{
  "preferredPaymentType": "VISA_TOKEN",
  "expirationDate": "2031-12-10T11:04:33.842",
  "roles": [
    {
      "name": "ADMIN",
      "permissions": [
        "REQUEST_PAYMENT"
      ]
    }
  ]
}
```

3.2.5.5 Responses

Regarding error handling, all codes have the same pattern as the return object: code, type, message and details(optional)

Code (http)	Description
204	No Content
400	Parameters missing in request
401	Unauthorized
403	Forbidden
404	Not found
500	Unexpected errors not mapped to status codes mentioned above

It is possible to verify two practical examples: programmatic via script or visual through the graphical interface (feature in the menu - Manage Service – Update subscription)

Scripts: service_grant_delete.py

service_grant_update.py

3.2.6 Service Activation

This operation is used whenever the Platform wants to activate an service. Service must be in INACTIVE status.

3.2.6.1 Parameters

API: POST /services/{serviceId}/activation

serviced (string) - unique identifier of the service

3.2.6.2 Responses

Regarding error handling, all codes have the same pattern as the return object: code, type, message and details(optional)

Code (http)	Description
204	No Content
401	Unauthorized
403	Forbidden
404	Resource not found
409	Activation not allowed due to service's status or invalid DMN/triggers
500	Unexpected errors not mapped to status codes mentioned above

It is possible to verify two practical examples: programmatic via script or visual through the graphical interface (feature in the menu - Manage Service – Deactivate/Recativate)

Script: services_activation.py

3.2.7 Service Inactivation

This operation is used whenever the Platform wants to inactivate an service. Service must be in ACTIVE or EXPIRED status.

3.2.7.1 Parameters

API: DELETE /services/{serviceId}/activation

serviced (string) - unique identifier of the service

3.2.7.2 Responses

Regarding error handling, all codes have the same pattern as the return object: code, type, message and details(optional)

Code (http)	Description
204	No Content
401	Unauthorized
403	Forbidden
404	Resource not found
409	Activation not allowed due to service's status
500	Unexpected errors not mapped to status codes mentioned above

It is possible to verify two practical examples: programmatic via script or visual through the graphical interface (feature in the menu – Manage Service – Deactivate/Recativate)

Script: services_deactivation.py

3.3 Transactions History

3.3.1 Querying Transactions

This operation allows sending data in a reliable, verifiable and guaranteed way by the device to trigger a service.

3.3.1.1 Parameters

API: POST /transactions/list

x-jws-signature (string) - At the moment, we are expecting the DAB Account ID, so format is UUID. In the future it will be a JWS containing the DAB Account ID info

Example : 0af4014e-81b8-15af-8181-b94f71cf0001

3.3.1.2 Request body

```
{
  "deviceId": [
    "0e92e8dd-3d38-f3a2-1925-28bd968752a4"
  ],
  "startDate": "2023-03-12T03:18:08.448Z",
  "endDate": "2023-03-12T03:18:08.448Z",
  "pageNumber": 1,
  "pageSize": 20,
  "sortField": "IMSI",
  "sortFieldOrderDirection": "ASC"
}
```

3.3.1.3 Responses

Regarding other error handling, all codes have the same return object pattern: code, type, message and details (optional)

Code (http)	Description

200

Ok


```
{
  "totalResults": "Unknown Type: long",
  "pageNumber": 0,
  "pageSize": 0,
  "sortField": "IMSI",
  "sortFieldOrderDirection": "ASC",
  "content": [
    {
      "recordedTime": "2023-03-12T03:18:08.449Z",
      "identifier": "277890bf-dd90-3346-3a9b-b1cfe4a95809",
      "processingStatus": "IN_PROGRESS",
      "outcome": "NO_ERRORS",
      "service": {
        "identifier": "48bb945a-7975-c3e6-518a-4e920a57507a",
        "status": "ACTIVE",
        "name": "string",
        "triggers": [
          {
            "name": "string",
            "type": "UUID",
            "value": "string"
          }
        ],
        "businessLogic": "string",
        "expirationDate": "string",
        "acceptedPaymentMethods": [
          {
            "paymentType": "VISA_TOKEN"
          }
        ],
        "serviceOwnerDabAccountId": "23df12f0-43f8-da7e-35a7-fe3d7903
d57d"
      },
      "triggerDeviceId": "385c0dfa-88f8-f990-9af1-f732edd24c1a",
      "targetDeviceId": "dbc5d25a-39ca-9bc6-c7bd-349bb93462ee",
      "basicOutputs": [
        {
          "name": "string",
          "errorMessage": "string",
          "value": "string"
        }
      ],
      "messageOutputs": [
        {
          "name": "string",
          "errorMessage": "string",
          "recipient": "b66025d7-d207-f3cd-9db2-69adae2048e1",
          "method": "string",
          "message": "string"
        }
      ],
      "messageToUserOutputs": [
        {
          "name": "string",
          "errorMessage": "string",
          "recipient": "string",
          "notificationBody": "string",
          "notificationTitle": "string",
          "notificationType": "string",
          "process": "string"
        }
      ],
      "offChainOutputs": [
        {
          "name": "string",
```

```
          "errorMessage": "string",
          "stepResponse": {
            "content": "string",
            "identifier": "string",
            "status": "string"
          }
        }
      ],
      "triggers": [
        {
          "name": "string",
          "type": "UUID",
          "value": "string"
        }
      ]
    }
  ]
}
```


401	Unauthorized
403	Forbidden

3.3.1.4 Parameters

API: GET /transactions/{transactionId}

x-jws-signature (string) - At the moment, we are expecting the DAB Account ID, so format is UUID. In the future it will be a JWS containing the DAB Account ID info

Example : 0af4014e-81b8-15af-8181-b94f71cf0001

3.3.1.5 Responses

Code (http)	Description
200	Ok


```
{
  "recordedTime": "2023-03-12T03:26:44.104Z",
  "identifier": "c48238d8-ecd4-bd3b-6c14-15e752558535",
  "processingStatus": "IN_PROGRESS",
  "outcome": "NO_ERRORS",
  "service": {
    "identifier": "99797b1c-d1e5-08c6-b957-5fb5a5098b49",
    "status": "ACTIVE",
    "name": "string",
    "triggers": [
      {
        "name": "string",
        "type": "UUID",
        "value": "string"
      }
    ],
    "businessLogic": "string",
    "expirationDate": "string",
    "acceptedPaymentMethods": [
      {
        "paymentType": "VISA_TOKEN"
      }
    ],
    "serviceOwnerDabAccountId": "5f43e304-465f-e8db-d254-b138cd1ef0ae"
  },
  "triggerDeviceId": "4bae3f6c-c3c3-4b2c-5679-edec5c22d863",
  "targetDeviceId": "2e1175b5-27d8-9b54-e3e1-86f631779935",
  "basicOutputs": [
    {
```

```
      "name": "string",
      "errorMessage": "string",
      "value": "string"
    }
  ],
  "messageOutputs": [
    {
      "name": "string",
      "errorMessage": "string",
      "recipient": "860abdcc-c712-93f2-12b3-83d643cb7f7c",
      "method": "string",
      "message": "string"
    }
  ],
  "messageToUserOutputs": [
    {
      "name": "string",
      "errorMessage": "string",
      "recipient": "string",
      "notificationBody": "string",
      "notificationTitle": "string",
      "notificationType": "string",
      "process": "string"
    }
  ],
  "offChainOutputs": [
    {
      "name": "string",
      "errorMessage": "string",
      "stepResponse": {
        "content": "string",
        "identifier": "string",
        "status": "string"
      }
    }
  ],
  "triggers": [
    {
      "name": "string",
      "type": "UUID",
      "value": "string"
    }
  ]
}
```


401	Unauthorized
403	Forbidden

It is possible to verify two practical examples: programmatic via script or visual through the graphical interface (feature in the menu - Transactions)

Scripts: transaction_details.py

transaction_list.py

4. Functional Flows

4.1 Onboarding a new device

The Onboarding of a new device consists of registering and verifying it on the platform, so that it can take advantage of the capabilities provided by the same: Identity, Security, Traceability, allowing to associate this device to services and to transact with all elements within the platform. To successfully onboard a new device, a few steps are required as shown below:

1. Initially, the device must be registered (If necessary, check item 2.1.1). A Device registration is the moment when the device starts to be recognized by the DAB, where the device identity is created and the DAB has access to its data. In this way, it is possible to perform any action related to the device
2. After completing the registration, the device is created on the platform, but it does not yet belong to a specific account, so it is necessary to associate it. (check item 3.1.1). The device association to an account occurs in 2 ways: via API (device assignment) and automatic association. Via API, the account that the device will be associated is the same linked to the authentication credentials. In LabKit this is the available association. The automatic association allows that in the registration process, when registering a device, the credentials used during that registration already indicate to which account the device must be associated with (will be available soon).
3. After association, a confirmation message and an automatic validation flow between the DAB and the device is started. You can follow the evolution of your device's status through the Status in the Decive Journey (Item 2.2), or through the Platform Journey (Item 3.1.4)

4.2 Creating and provisioning a new Service (Service Wizard)

DAB is a platform that makes it possible to create use cases that allow you to create your own business logic that is called a Service. The Use case will contemplate the interaction between the devices/entities registered in the platform, theses service and External Applications.

Services are business logic that will be executed, defined by the user through a low-code tools that the DAB makes available and these business logics are executed within the DAB and DLT, that is, what allows having a business logic that connects the elements is the service, who sets the rules.

When a device performs a trigger, it sends a request to the server for a certain service with a set of parameters (triggers). This Request is signed on the Device and validated on DAB and DLT to confirm the Device identity, therefore, when this set of triggers are sent, the service receives it, proceeds and defines which action will be performed.

When the service is created (entity), it's start as Inactive Service and it is necessary to activate it (in the same way, you can deactivate or update the service – alternative flows)

Once the service is active, the device has to subscribe to the service. Each device subscription generate a Grant to this service on Device Entity.

Grant:

Grant is a permission to a Device Access a specific service. It's created as result of a device subscribes to a service. A Grant for that service is generated within the entity/device with an expiration date and payment method.
The Grant can also be updated (Update Grant) or deleted if you want to terminate the subscription (Remove Grant)

If you intend to deactivate the service, the service is completely inactive. When a device service is disabled, the Grant is removed (deactivate the service for the device)

4.3 Executing a Service

To run the service, the device must already be registered, the service created and an active subscription.

The execution of the service is the moment when the device executes the action in relation to the service or requests the execution of an action from the service.

The business transactions (business flow transaction) are the device's way of triggering the execution of the service, that is, it is an action in which the device receives the parameters of the person responsible for the use case app action and, based on its identity, signs these information and sends it to the DAB which records it immutably in the DLT.

Subsequently, value transactions can be generated that involve payments, or not, depending on the action that was configured within that service. The same service can support several actions according to the parameters it receives.

For example, let's think of a parking use case, there are 3 simple actions that a device performs in parking: start session, end session and execute the payment. Within the same service that is parking, we can have the 3 actions. According to the parameters, it is possible to inform if it is a start session, if it is a end session and based on that result make/confirm the payment.

4.4 Checking History by device or service

When an action is executed on a device, it is registered in the DLT and this process allows checking whether it was executed or not.

That is, it allows you to consult the transaction history and see what was done on the device or the service (depending on the filters).

For example, if it is a service manager, it will check which devices consumed that service, in turn, a device manager will check which services the device accessed.

It is possible to consult various information, namely: Recorded Time, Identifier, Status, Outcome, payment type, among others.

That is, it allows you to consult the transaction history and see what was done on the device or service (depending on the filters).

5. Appendix

Installing the SDK

To install the SDK, download the zip file (SDK + installation script), unzip and run the script.
Copy the SDK package and the installation script extract-sdk.sh to the /home/pi folder and extract the SDK using the following command:
bash extract-sdk.sh dab-sdk-Linux-armv7l-[BUILD-DATE-TIME].tar.gz
When executing the set-up script, it will prompt to define the credentials of the SDK provided by Vodafone.

For informational purposes, the default installation will install the SDK to the /opt folder and automatically perform the following actions:

1. Creates an dab-sdk.conf file under the /etc/ld.so.conf.d folder to point to the DAB shared libraries.
2. Sets up a cron job to start up the dab-sms-handler every minute via the dab-sms-handler-sheduler.sh script.
3. Sets up a startup script (on /etc/rc.local) to call the dab-app-wrapper with the registration option at device boot up. **On SDK Labkit version, this option is disabled to allow developers define when start the device registration.**
4. The startup script also is responsible to call the modem initialization script, connecting your USB modem to network.
5. Installs the required build tools snapd, build-essential, g++, libssl-dev, libcpputest-dev, git, default-jdk, rapidjson-dev, and cmake.
6. Reboot to complete the installation.

At the end of the installation, you will be asked for the customer ID provided by Vodafone.

Note: If necessary, it's possible to redefine the credentials of the SDK provided by Vodafone – command for the credentials:

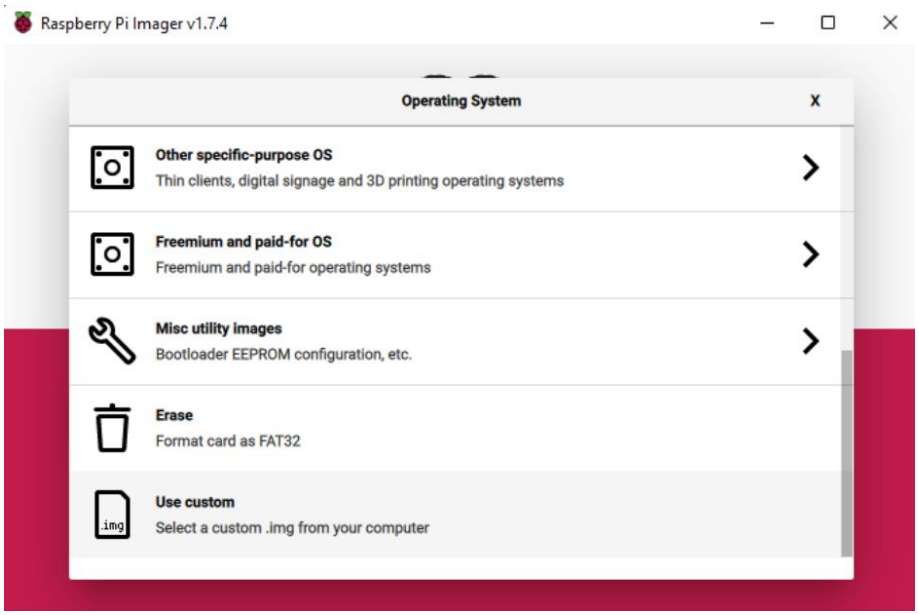
```
pi@raspberrypi4:/opt/dab-sdk $ bash set-dab-credentials.sh
```

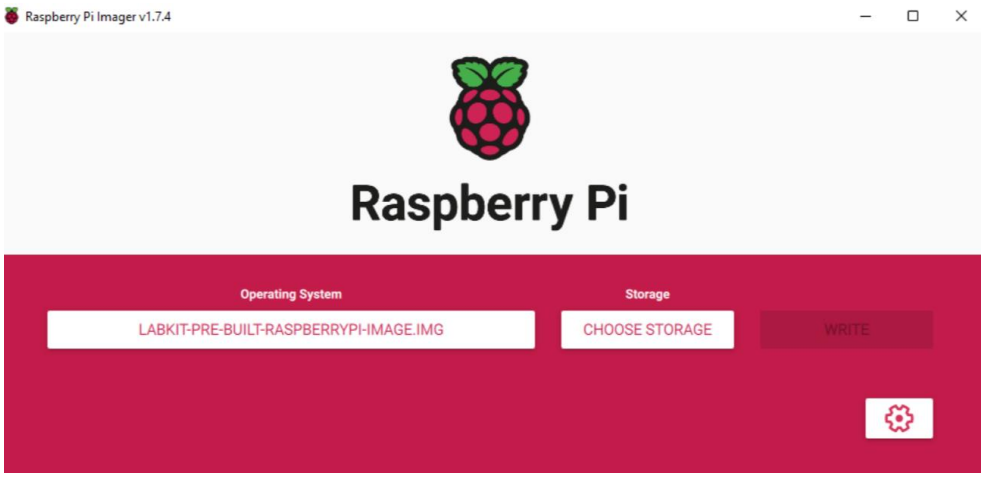
Installing the Img

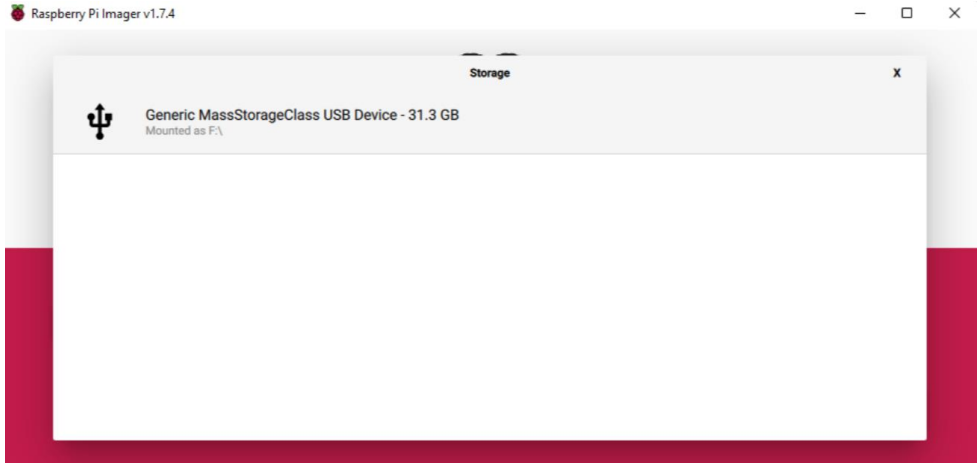
For preconfigured DAB image installation, download the raspberry PI image application (https://www.raspberrypi.com/software/)

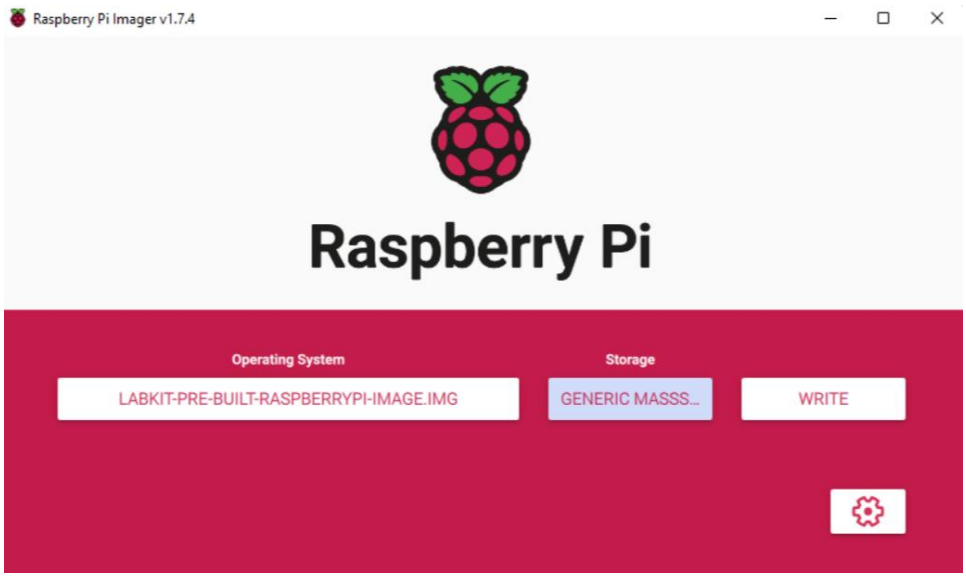
Once the application is installed, it is necessary to download the image from the repositor.

In the operating system, you must select the Raspberry PI image downloaded and in storage is where the image will be recorded - SD card.



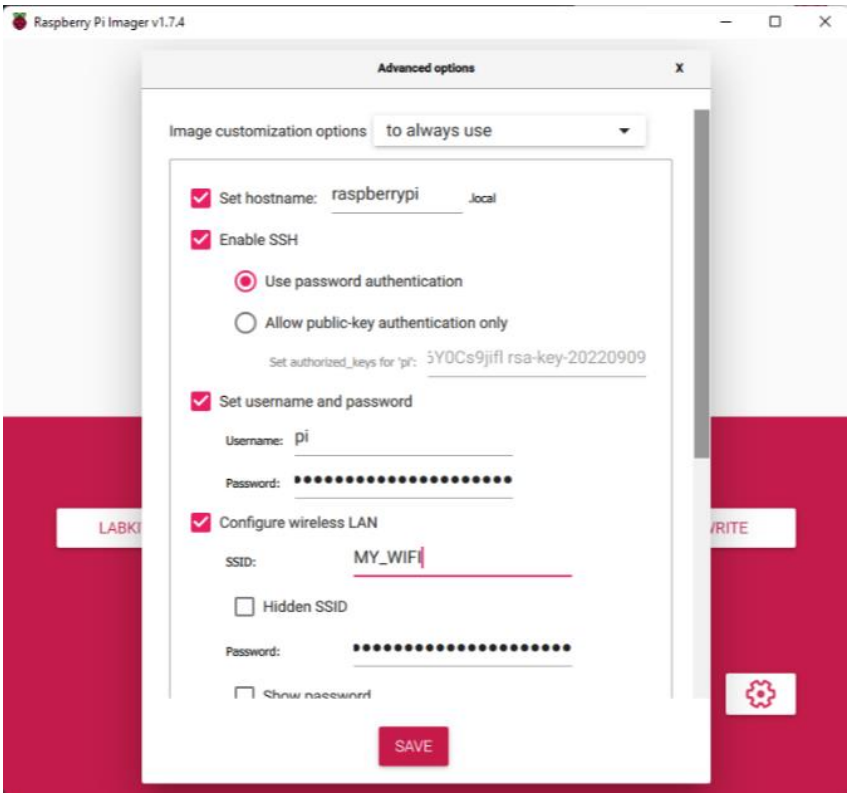






The settings allow you to define parameters for the image, in case you want your image to be connected to your Wifi, to make it easier to use without having to access the PI directly for this. In this way you can do an automated configuration of the:

- PI name (hostname)
- User (default: pi password: pi)



After validating the settings, just click write and wait for the process to run. Once the image has been recorded on the SD Card, insert the SD card in the PI, connect the PI to the electrical network and then use the graphical interface to configure the DAB access credentials provided by Vodafone

The Raspberry will be running, if it is configured by your wi-fi you can access its IP by the url, otherwise access it locally.

Installing the Device Webapp

Labkit - Device Webapp was thought to enhancing your user experience, we have provided an unique visual performance called Web app, where it allows you to interact with device features through your browser running locally on your own Laptop, over a simple Tornado Web Server.

Setup

Make sure you have python 3.9 (or python 3.10) and Python PiP installed.

You can download python directly from Pythoh webpage (https://www.python.org/downloads/), or alternatively you can use your OS Package Manager (APT, RPM, chocolatey...)

Example:

```
# Update packages references
apt update -y


#Installing Python and dependencies
apt install python3 python3-pip pipenv -y
```

Install Locally

1. Extract the zip file content.
2. On command line, install the python modules required:

```
# Installing Tornado Web Server Module
pip install Tornado
# Installing Regex Module
pip install regex
# Installing Requests Module
pip install requests
```

*** On debian bases OS, you can run the install script directly:

```
sudo bash install.sh
```

Start Application

After Python3 and modules installed, you can run your WebApp, according the follow examples:

```
# Running on default config
python3 webapp.py
```

```
# Running on verbose mode
python3 webapp.py -v

# Running on specific port
python3 webapp.py -p [PORT]

# Running on Help options
python3 webapp.py -h
```

After the startup, you can access the WebApp directly on your browser http://[HOST]:[PORT] (Default: http://localhost:8888).

***Access the Webapp from remote machine it's possible since the port is exposed to remote access.

Confire your credentials

When you connect on your WebApp, it's necessary to configure the Access token and DAB Account to be used on DAB requets. You can to it access the "Authorize" button on the Top-Right corner.

Installing the Platform Webapp

Labkit - Platform Webapp was thought to enhancing your user experience, we have provided an unique visual performance called Web app, where it allows you to interact with device features through your browser running locally on your own Laptop, over a simple Tornado Web Server.

Setup

Make sure you have python 3.9 (or python 3.10) and Python PiP installed.

You can download python directly from Pythoh webpage (https://www.python.org/downloads/), or alternatively you can use your OS Package Manager (APT, RPM, chocolatey...)

Example:

```
# Update packages references
apt update -y

#Installing Python and dependencies
apt install python3 python3-pip pipenv -y
```

Install Locally

1. Extract the zip file content.
2. On command line, install the python modules required:

```
# Installing Tornado Web Server Module
pip install Tornado
# Installing Regex Module
pip install regex
# Installing Requests Module
pip install requests
```

*** On debian bases OS, you can run the install script directly:

```
sudo bash install.sh
```

Start Application

After Python3 and modules installed, you can run your WebApp, according the follow examples:

```
# Running on default config
python3 webapp.py

# Running on verbose mode
python3 webapp.py -v

# Running on specific port
python3 webapp.py -p [PORT]

# Running on Help options
python3 webapp.py -h
```

After the startup, you can access the WebApp directly on your browser http://[HOST]:[PORT] (Default: http://localhost:8888).

***Access the Webapp from remote machine it's possible since the port is exposed to remote access.

Confire your credentials

When you connect on your WebApp, it's necessary to configure the Access token and DAB Account to be used on DAB requets. You can to it access the "Authorize" button on the Top-Right corner.