DOKUMENTACJA PROJEKTOWA INTERNET OF THINGS

Connection to the device (OPC UA server).

To connect to the the server i used library asyncua. At the start the application will ask for url to opcua server and connection string for devices.
It will be stored in config.ini

```
[opcua]
url = opc.tcp://localhost:4840/
```

After connection we create Factory instance (from factory.py) and get list of devices. For each device the application will create a instance of agent class (agent.py) and ask for connection string.

Agent Config

After providing the connection string, the agent will be created using the IoTHubDeviceClient.create_from_connection_string(self.connectionString) function located in the agent.py file.

The agent is responsible for sending data to Iot Hub. Sending is done in the telemetry() function.

Telemetry

```
async def telemetry(self):
    data = {
        "ProductionStatus": await self.device.get("ProductionStatus"),
        "WorkorderId": await self.device.get("WorkorderId"),
        "GoodCount":await self.device.get("GoodCount"),
        "BadCount":await self.device.get("BadCount"),
        "Temperature":  await self.device.get("Temperature"),
    }
    print(data)
    msg = Message(json.dumps(data), "UTF-8", "JSON")
    self.iotClient.send_message(msg)
```

The agent sends telemetry every 5 secounds to iotHub.

Thu Jan 05 2023 15:55:07 GMT+0100 (czas środkowoeuropejski standardowy):


```
{
  "body": {
    "ProductionStatus": 0,
    "WorkorderId": "0bd3c64b-8a00-46be-9419-da1eae538d91",
    "GoodCount": 664,
    "BadCount": 78,
    "Temperature": 25.84783384741981
  },
  "enqueuedTime": "Thu Jan 05 2023 15:55:07 GMT+0100 (czas
środkowoeuropejski standardowy)",
  "properties": {}
}
```

Device Twin

There are 2 functions integrating in device twin.

1.

```
async def patchTwinReportedProperties(self, prop):
  self.iotClient.patch_twin_reported_properties(prop)
```

```
tasks.append(
  asyncio.create_task(self.patchTwinReportedProperties({'DeviceError': await self.device.get("DeviceError")}))
)

tasks.append(
  asyncio.create_task(self.patchTwinReportedProperties({'ProductionRate': await self.device.get("ProductionRate")}))
)
```

This function is adding DeviceError and ProductionRate to reported properties of device twin.

Device twin


```
    },
    "modelId": "",
    "version": 1093,
    "properties": {
        "desired": {
            "ProductionRate": 15,
            "DeviceError": 0,
            "$metadata": {
                "$lastUpdated": "2023-01-05T14:35:09.0364884Z",
                "$lastUpdatedVersion": 25,
                "ProductionRate": {
                    "$lastUpdated": "2023-01-05T14:35:09.0364884Z",
                    "$lastUpdatedVersion": 25
                },
                "DeviceError": {
                    "$lastUpdated": "2023-01-05T14:35:09.0364884Z",
                    "$lastUpdatedVersion": 25
                }
            },
            "$version": 25
        },
        "reported": {
            "LastMaintenanceDate": "2023-01-05T15:50:19.952782",
            "DeviceError": 1,
            "ProductionRate": 45,
            "WorkorderId": "43865a7b-7572-4d4a-af0b-c933fc206c4b",
            "MaintenanceDoneDate": "05/01/2023 12:20:01",
            "$metadata": {
                "$lastUpdated": "2023-01-05T14:56:25.3077447Z",
                "LastMaintenanceDate": {
                    "$lastUpdated": "2023-01-05T14:50:21.2894512Z"
                },
                "DeviceError": {
                    "$lastUpdated": "2023-01-05T14:56:25.1514261Z",
                    "errors": {
                        "$lastUpdated": "2022-12-24T11:19:56.7064717Z"
                    },
                    "errorCode": {
                        "$lastUpdated": "2022-12-24T11:19:56.7064717Z"
                    }
                },
                "ProductionRate": {
                    "$lastUpdated": "2023-01-05T14:56:25.3077447Z"
                },
                "WorkorderId": {
                    "$lastUpdated": "2023-01-05T11:43:53.4578816Z"
                },
                "MaintenanceDoneDate": {
                    "$lastUpdated": "2023-01-05T11:20:02.0266862Z"
                }
            },
            "$version": 1068
        }
    },
    "capabilities": {
```

2.

```python
def handleRequestRecives(self, request):

  if request.name == "ResetErrors":
    print('Reseting errors...')
    node = self.device.client.get_node(self.device.id)

    self.tasks.append(
      node.call_method("ResetErrorStatus")
    )
    self.iotClient.send_method_response(MethodResponse(request.request_id, 0))

  elif request.name == "Stop":
    print('Emergency stop...')
    node = self.device.client.get_node(self.device.id)
    self.tasks.append(
      node.call_method("EmergencyStop")
    )
    self.iotClient.send_method_response(MethodResponse(request.request_id, 0))

  elif request.name == "Maintenance":
    print("Adding maintenance date...")
    self.iotClient.patch_twin_reported_properties({"MaintenanceDate":  datetime.datetime.now().isoformat()})

    self.iotClient.send_method_response(MethodResponse(request.request_id, 0))

  else:
    print('Method not found!')
```

The handleRequestRevies function is used to send direct methods.
ResetErrors will reset all errors on device

Stop will run emergency stop on device

Maintance will patch twin reported properties and add there maintanceDate

Buisness Logic

Business logi cis made by queries and fucntions that are stored in asa and functions folder