Send action to ARTIK device

ARTIK Lecture 3
CONTENTS

Our class plan

Overall structure of example

Make your own sensor

Send action to ARTIK device

Assignment
Our class plan
Our class plan

- **5/9**
  Analyze sample codes. Getting data from ARTIK (not real time)

- **5/16**
  Getting data from ARTIK (real time)

- **5/23**
  Ordering devices or sensors to do any action.

- **5/30**
  Custom sensor, Theoretical knowledges

- **6/6**
  Final presentation
Before start

- You should change "AUTH REDIRECT URL" in application tap. [https://developer.artik.cloud/dashboard](https://developer.artik.cloud/dashboard)
Previous assignment
onPing()

- Called in each 30 second if the websocket is connected.

- Monitor onPing() method. If it is not called for 30 second from open or previous ping, we can recognize the disconnection.

- In almost socket (not only ARTIK but also other overall sockets), onClose() is called when only user close socket "normally".

- We cannot recognize socket disconnection automatically without sending or receiving data.
Overall structure of example
Today’s goal

- We make our own abstract temperature sensor.
- This sensor automatically send random temp at each 1 second.
- We can turn on/off our own temperature sensor remotely.
Overall structure

1. **Actions**: Turn device on/off
   - Send Action
   - Monitor data

2. **Actions**: Turn device on/off
   - Send Action
   - Send data

3. **State**: Report Temp data
   - STATE

4. **State**: Report Temp data
   - STATE

**ANDROID**

**My Temp Device**

*Image: Samsung ARTIK Cloud - The Data Exchange Platform for IoT Developers*
WebSockets in ARTIK

Firehose WebSocket

- To listen to messages sent by the source devices that the application monitors.

Device channel WebSocket

- To receive messages targeted to your applications or devices.
- To send action to targeted device.
- Allows the applications or devices to send messages back to ARTIK.
Which API we use?

1. **Device channel (application)**
   - **Actions**: Turn device on/off
   - **API**: Firehose(app)
   - **Function**: Report Temp data

2. **Device channel (device)**
   - **Actions**: Turn device on/off
   - **API**: REST API
   - **Function**: Report Temp data

3. **State**
   - **Function**: Report Temp data

4. **State**
   - **Function**: Report Temp data

Android

My Temp Device
Warning: Aggregated structure

- In this tutorial, android application and own sensor device is aggregated in one application.

- So we use one Data channel websocket for both application and device. If you make independent sensor, you should make own data channel websocket for both app and sensor.
Application part & Sensor part

• Application Part
  – Send turn on/off action through data channel websocket(app) when button clicked.
  – Monitor data in real time through firehose websocket.
  – Show data in real time.

• Sensor part
  – If turn on, send random temp value at each 1 second. (Rest API)
  – Receive action through data channel websocket(sensor).
  – Check action whether it is turn on or turn off.
  – Change state according to it.
Make your own sensor
Go to ARTIK Dashboard/Device type

- [https://developer.artik.cloud/dashboard/devicetypes](https://developer.artik.cloud/dashboard/devicetypes)
- Click "New device type" button
Write device name

- Write both name as what you want.
- However Unique name should be unique in ARTIK cloud.
- Click "Create Device Type"
Click ‘New manifest’ button

Create a manifest for MyDevice

ARTIK Cloud is designed to communicate with any device regardless of how data is structured. The Manifest provides a way for you to describe your data, so that you can start sending data to ARTIK Cloud.

Manifest is a kind of recipe of your device.
It shows some properties such as what data your sensor will send.

Device Manifest
- Dive into the details »
- Follow a step by step guide »

Your Data
- Sending and receiving data »
- Keep the data flowing with Web Sockets »
Define device field

- You can define what kind of data will be sent by your sensor.
- Save and click 'next: device actions' button
Define device action

- You can define what your device can do.
- Memory the name of action such as "turnon" and "turnoff"
- Save and click 'Next: activate manifest' button
Check your manifest and activate it

Device Fields
Describe fields for each piece of data produced by this device.

Device Actions
Describe actions that this device is capable of receiving.

Activate Manifest
Publish this device manifest on the ARTIK Cloud platform.

Your manifest is ready to be activated and does not require approval before going live. Activating this manifest will replace the current manifest. The device type will stay private.

Fields
- temp
- Double
- SI.CELSIUS

Actions
- turnon
- Action
- turnoff
- Action

Field name: used when we send data
Action name: used when we send action
Connect your new device

ARTIK Cloud works with many smart device types – start typing to find yours.

Connect another device

Find your device which you made.
If you make your own data type, you can find the device here.
This device is shown in only your account.
## Memory device ID and device token

<table>
<thead>
<tr>
<th>Device Info</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DEVICE ID</strong></td>
</tr>
<tr>
<td><strong>DEVICE TYPE ID</strong></td>
</tr>
<tr>
<td><strong>NAME</strong></td>
</tr>
<tr>
<td><strong>DEVICE TOKEN</strong></td>
</tr>
<tr>
<td><strong>REVOKE TOKEN</strong></td>
</tr>
</tbody>
</table>
Set your application permission

- Find your device which you made.
- Turn on read and write permission to your application.
Send action to ARTIK device
Get project

• Get today’s example project in comnet.

• This tutorial is similar to previous tutorial which is used in last class.

• Basic layout xml and onCreate/onResume/onPause function is written.

• We will write other main process in this class.
See some pre-written code

```java
private Button openFirehoseWebSocketButton;
private Button openDeviceChannelWebSocketButton;
private Button turnOnButton;
private Button turnOffButton;
private TextView mWelcome;
private TextView listenedText;

private FirehoseWebSocket mFirehoseWebSocket;
private DeviceChannelWebSocket mDeviceChannelWebSocket;

private boolean turnon;
private SensorTemp sensorTemp;
```

- FirehoseWebSocket & DeviceChannelWebSocket both are used.

- Only one DeviceChannelWebSocket is used. It worked as not only app’s websocket but also sensor’s websocket. Because this example aggregate app and sensor.

- Boolean ‘turnon’ saves the state of sensor.
See some pre-written code

```java
public static final String KEY_ACCESS_TOKEN = "Access_Token";
private static final String DEVICE_ID = "f2ec75c5e926454d866bf5bcc0267641";
private static final String SOURCE_DEVICE_ID = "f2ec75c5e926454d866bf5bcc0267641";
private static final String CLIENT_ID = "7ff692287e5f4d5294414c0a4c51baa9";
private static final String DEVICE_TOKEN = "b7538a09fe2a49409dc094a05cf7df25";
private static final String ACTION_NAME_TURN_ON = "turnon";
private static final String ACTION_NAME_TURN_OFF = "turnoff";
```

**DEVICE_ID**: sensor's device id  
**SOURCE_DEVICE_ID**: application's device id (or name)  
Name of object who send action.  
**CLIENT_ID**: application id (different to SOURCE_DEVICE_ID)  
**DEVICE_TOKEN**: device token of sensor device.  
**ACTION_NAME**: action names which you define.
See some pre-written code

```java
turnOnButton = (Button) findViewById(R.id.action_turnon);
turnOnButton.setOnClickListener(v -> {
    Log.v(TAG, "turn on button is clicked.");
    SendActionInBackground sendAction = new SendActionInBackground();
    sendAction.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR, ACTION_NAME_TURN_ON);
});

turnOffButton = (Button) findViewById(R.id.action_turnoff);
turnOffButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Log.v(TAG, "turn off button is clicked.");
        SendActionInBackground sendAction = new SendActionInBackground();
        sendAction.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR, ACTION_NAME_TURN_OFF);
    }
});
```

- Define turn on/off buttons click listener.
- We will make "SendActionInBackground" class which extend AsyncTask.
See some pre-written code

```java
openFirehoseWebSocketButton = (Button) findViewById(R.id.button_openhws);
openFirehoseWebSocketButton.setOnClickListener(new View.OnClickListener()) {
    @Override
    public void onClick(View v) {
        try {
            connectFirehoseWebSocket();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
});
openFirehoseWebSocketButton.setClickable(false);

openDeviceChannelWebSocketButton = (Button) findViewById(R.id.button_opendws);
openDeviceChannelWebSocketButton.setOnClickListener(new View.OnClickListener) {
    @Override
    public void onClick(View v) {
        try {
            connectDeviceChannelWebSocket();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
});
openDeviceChannelWebSocketButton.setClickable(false);
```

- Define turn on/off buttons click listener.
- We will make "connectDeviceChannelWebSocket" method.
- FirehoseWebSocket is already made as last class.
See some pre-written code

```java
@override
protected void onPause() {
    super.onPause();
    if (sensorTemp != null) {
        sensorTemp.cancel(true);
    }
}

@override
protected void onResume() {
    super.onResume();
    if (sensorTemp == null) {
        sensorTemp = new SensorTemp();
        sensorTemp.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR);
    }
}
```

- SensorTemp is an AsyncTask which send temp data at each 1 second.
- OnResume, we execute it and OnPause, stop it.
Connecting Device channel websocket

Almost structure is similar to Firehose websocket

Register source device (who send action) when websocket is open.

In both app and sensor, we should register who will send action.

```java
private void connectDeviceChannelWebSocket() throws Exception {
    OkHttpClient client = new OkHttpClient();
    client.setRetryOnConnectionFailure(true);

    mDeviceChannelWebSocket = new DeviceChannelWebSocket(false, client, new ArtikCloudWebSocketCallback()) {
        @Override
        public void onOpen(int i, String s) {
            Log.e(TAG, "DeviceChannelWebSocket: onOpen");

            RegisterMessage registerMessage = new RegisterMessage();
            registerMessage.setAuthorization("bearer " + DEVICE_TOKEN);
            registerMessage.setCid(CLIENT_ID);
            registerMessage.setDid(SOURCE_DEVICE_ID);

            try {
                Log.e(TAG, "DeviceChannelWebSocket: onOpen: registering " + DEVICE_ID);
                mDeviceChannelWebSocket.registerChannel(registerMessage);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        @Override
        public void onMessage(MessageOut messageOut) {
            Log.e(TAG, "DeviceChannelWebSocket: onMessage(" + messageOut.toString() + ")");
        }
    }
```
OnAction() is called when device "RECEIVE" an action.

So this is sensor part. When sensor receive an action, it check whether the action is turn on or turn off.
Send temp data at each 1 second

Extend AsyncTask

Sleep 1 second

Sensor’s role

If the sensor is turn on it send temp data
Send temp data at each 1 second

- Send data as you learned in previous lecture.
- Put data as your defined device field like key "temp"
- Send data with "Sensor's device id"

```java
private void sendTemp(double temp) {
    final String tag = TAG + " sendTempActionAsync";

    Map<String, Object> data = new HashMap<String, Object>();
    data.put("temp", temp);

    MessageAction msg = new MessageAction();
    msg.setKeyId(DEVICE_ID);
    msg.setDate(data);

    try {
        MessagesApi.sendMessageAsync(msg, new ApiCallback<MessageEnvelope>() {
            @Override
            public void onFailure(ApiException exc, int i, Map<String, List<String>> stringListMap) {
                processFailure(tag, exc);
            }
            @Override
            public void onSuccess(MessageEnvelope result, int i, Map<String, List<String>> stringListMap) {
                Log.v(tag, "onSuccess response to sending message = " + result.getData().toString());
            }
            @Override
            public void onDownloadProgress(long bytes, long contentLen, boolean done) {
            }
            @Override
            public void onUploadProgress(long bytes, long contentLen, boolean done) {
            }
        });
    } catch (ApiException exc) {
        processFailure(tag, exc);
    }
}
```
Send action to ARTIK device

- Send action with following way.
- Put source device id (who send action) and device id (who receive action).
- Sdid : source device id
  Ddid : destination device id
- Put action name as you defined.

```java
private void sendActionInDeviceChannelWS(String actionName)
{
    ActionIn actionIn = new ActionIn();
    ActionDetails actionDetails = new ActionDetails();

    ArrayList<ActionDetails> actions = new ArrayList<>();
    ActionDetailsArray actionDetailsArray = new ActionDetailsArray();

    actionDetails.setName(actionName);
    actions.add(actionDetails);
    actionDetailsArray.setActions(actions);
    actionIn.setActionDetailsArray(actionDetailsArray);
    actionIn.setSdid(SOURCE_DEVICE_ID);
    actionIn.setCid(CLIENT_ID);
    actionIn.setDid(DIDevice_ID);
    actionIn.setTs(System.currentTimeMillis());

    try{
        mDeviceChannelWebSocket.sendAction(actionIn);
        Log.d(TAG, "DeviceChannelWebSocket sendaction:" + actionIn.toString());
    } catch (Exception e){
        e.printStackTrace();
    }
}

private class SendActionInBackground extends AsyncTask<String, Void, Void>{
    @Override
    protected Void doInBackground(String... params) {
        sendActionInDeviceChannelWS(params[1]);
        return null;
    }
}
```
Overall flow

Sensor sending data

Send data to ARTIK with REST API

onAction() in sensor’s data channel websocket

ARTIK

Send action turn on / off in data channel websocket

App monitoring data

onMessage() in app’s firehose websocket
Assignment
Final presentation and report

- Details will be attached until 5/31 in comnet homepage.

- We will have presentation class at 6/6 10:00AM.

- Handle "What you will make" not "What you have made"

- Final report is very important. You should write your implementation requirements and plans in detail.

- Your final project will be evaluated according to this report.
Submit

- Submit your final report and presentation slide at Icampus.

- Submission date: Until 16.06.05 23:59

- Just submit one report and presentation slide for one team. Leader of each team should submit it.
THANK YOU!!!