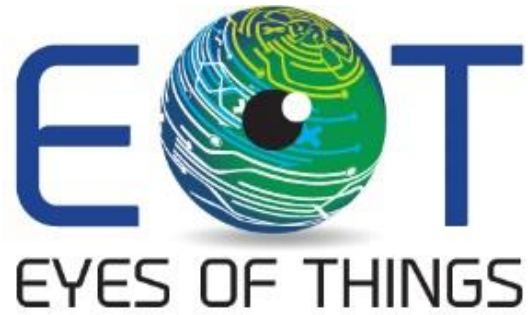
This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No 643924



D3.2

Android middleware API reference documentation

1 DOCUMENT INFORMATION

Deliverable Number	D3.2
Deliverable Name	Android middleware API reference documentation
Authors	Ruben Reiser (DFKI), Stephan Krauß (DFKI)
Responsible Author	Ruben Reiser (DFKI) e-mail: Ruben.Reiser@dfki.de phone: +49 361 205 75 3620
Keywords	MQTT, broker, API, Java
WP	WP3
Nature	R
Dissemination Level	PU
Planned Date	31.01.2016
Final Version Date	1.02.2016
Reviewed by	Alain Pagani (DFKI), O. Deniz (UCLM), E. Roche (THALES)

2 DOCUMENT HISTORY

Person	Date	Comment	Version
Stephan Krauß	21.01.2016	Initial version	0.1
Ruben Reiser	27.01.2016	Completed the UI documentation	0.2
Oscar Deniz	27.01.2016	Review	0.3
Elodie Roche	1.2.2016	Review	0.4

3 ABSTRACT

The Eyes of Things (EoT) project envisages a computer vision platform that can be used both standalone and embedded into more complex artefacts, particularly for wearable applications, robotics, home products, surveillance etc. The core hardware will be based on a number of technologies and components that have been designed for maximum performance of the always-demanding vision applications while keeping the lowest energy consumption.

An important functionality is to be able to communicate with other devices that we use every day. In EoT, a middleware is developed to allow configuration and basic control of the device from an external computer like a desktop/laptop PC or a tablet/smartphone. The wireless communication on which this middleware is based is additional to the existing wired debug capability of the Myriad SoC.

Apart from low-power hardware components, an efficient wireless communication protocol is necessary. Text-oriented protocols like HTTP are not appropriate in this context. Instead, the lightweight publish/subscribe message-based MQTT protocol was selected. With MQTT the typical scenario is that of a device that sends/receives messages, the messages being forwarded by a cloud-based message broker. In the EoT project we propose a novel approach in which each EoT device acts as an MQTT broker instead of the typical cloud-based architecture. This eliminates the need for an external Internet server, which not only makes the whole deployment more affordable and simpler but also more secure by default.

This document describes the Android app implementing the EoT middleware API.

4 TABLE OF CONTENTS

5 EOT MIDDLEWARE

The EoT middleware provides functionality for communication, control and configuration of EoT devices. In particular this functionality is implemented by Pulga, a tiny MQTT broker for EoT devices. A detailed description is available in the desktop middleware API reference documentation (D3.1). The Android client described in this document is communicating with the EoT device through this broker and implements the same Java API as the desktop client application.

6 EOT CONFIGURATION AND CONTROL APP FOR ANDROID

This section describes the counterpart of Pulga for mobile devices running Android. An MQTT [1][2] client can act as a publisher, a subscriber or both. Due to the small resources needed by the MQTT protocol, an MQTT client may run in any device from a micro controller up to a server. Basically any device that has a TCP/IP stack can use MQTT over it through:

- A plain TCP socket
- A secure SSL/TLS socket

The MQTT application only requires an MQTT library that connects the client with the broker through a network connection in order to send and receive small messages. There are many open-source MQTT client libraries available for a variety of programming languages such as Java, JavaScript, C, C++, C#, Go, iOS, .NET, Android, or Arduino.

The EoT Android MQTT client has been developed in Java using the Paho Java Client library [3]. The desktop and Android apps are both written in Java and share the same code basis.

6.1 Paho Java Client

The Paho Java Client [3] is an MQTT client library written in Java for developing applications that runs on the Java Virtual Machine, JVM. Moreover, it can be used under Android through the Paho Android Service.

Paho provides two APIs: MqttAsyncClient and MqttClient.

- MqttAsyncClient provides a fully asynchronous API where completion of activities is notified via registered callbacks.
- MqttClient is a lightweight client that blocks the application until an operation is complete. This class implements the blocking IMqttClient client interface.

The EoT MQTT application is divided into two packages: the "de.dfki.av.eotcontrolapp" and the "de.dfki.av.eotcontrolapp.ui". The first one contains classes that define the graphical user interface and user interaction. The second one contains classes that define the application logic. The "MQTT_Client" class manages the Paho client and provides all the functionalities needed.

6.2 API Specification

Class EoT_MQTT_Client

This class is an implementation of the EoT MQTT client.

1 Declaration

public class EoT_MQTT_Client
extends java.lang.Object

2 Fields

public final java.lang.String **topicEOTConnectToAP**
public final java.lang.String **topicEOTContentSD**
public final java.lang.String **topicEOTCreateAP**
public final java.lang.String **topicEOTDeleteDirSD**
public final java.lang.String **topicEOTDeleteFileSD**
public final java.lang.String **topicEOTDisconnectFromAP**
public final java.lang.String **topicEOTDownloadFileSD**
public final java.lang.String **topicEOTGetDate**
public final java.lang.String **topicEOTListFilesSD**
public final java.lang.String **topicEOTMakeDirSD**
public final java.lang.String **topicEOTUpdateDate**
public final java.lang.String **topicEOTUploadElf**
public final java.lang.String **topicEOTUploadFileSD**
public final java.lang.String **topicSnapshot**

3 Constructor summary

EoT_MQTT_Client(String,int) Gets an instance of EoT_MQTT_Client

4 Method summary

- **askSnapshot()**
  Sends a message in the topicSnapshot topic to get the image from the broker
- **connect()**
  Connects the client to the MQTT server
- **connectionLost(Throwable)**
- **connectToAP(String, String, String)**
  Connects the EoT device to an external AP
- **createAP(String, String, String, String)**
  Creates a new AP configuration profile
- **createFolder(String)**
  Makes a new folder in the SD card
- **deliveryComplete(IMqttDeliveryToken)**
- **disconnect()**
  Disconnects the client
- **downloadFile(String, String)**

Downloads a file from the SD card

- **getDate()**
  Gets the current EoT device time/date
- **getFileSystemStructure(String)**
  Gets the paths of the SD card content
- **isConnected()**
  Checks if the client is connected
- **messageArrived(String, MqttMessage)**
- **publish(String, int, byte[])**
  Publishes / sends a message to an MQTT server
- **removeAll(String)**
  Removes a folder and its content recursively
- **removeContent(String)**
  Removes the content of a folder (or the SD card if /mnt/sdcard is used)
- **removeFile(String)**
  Removes a file from the SD card
- **resetAPConfig()**
  Resets the AP configuration to the default profile
- **setMainFrame(EoT_MainFrame)**
  Sets the main frame where results are displayed
- **subscribe(String, int)**
  Subscribes the client to a topic on an MQTT server
- **unsubscribe(String)**
  Unsubscribes the client from a topic
- **updateDate(String, String, String, String, String, String)**
  Changes the EoT device time/date
- **uploadFile(String, String)**
  Sends a file to the SD card

6 Constructor

public EoT_MQTT_Client(java.lang.String brokerip , int brokerport)

- Description
  Gets an instance of EoT_MQTT_Client

- Parameters
  i. brokerip – IP where the broker is running
  ii. brokerport – port used by the broker

7 Methods

- **askSnapshot**

public javax.swing.ImageIcon askSnapshot() throws MqttException

- Description
  Sends a message in the topicSnapshot topic to get the image from the broker

- Throws

* MqttException

- **connect**

public void connect () throws MqttException

  - Description
    Connects the client to the MQTT server

  - Throws
    * MqttException

- **connectionLost**

  - Parameters
    * cause

  - See also
    public void connectionLost (java.lang.Throwable cause)
    * MqttCallback#connectionLost(Throwable)

- **connectToAP**

public void connectToAP(java.lang.String SSID, java.lang.String security, java.lang.String pass) throws MqttException

  - Description
    Connects the EoT device to an external AP

  - Parameters
    * SSID
    * security
    * pass

  - Throws
    * MqttException

- **createAP**

public void createAP(java.lang.String SSID, java.lang.String security, java.lang.String pass, java.lang.String channel) throws MqttException

  - Description
    Creates a new AP configuration profile

  - Parameters
    * SSID
    * security
    * pass
    * channel

  - Throws

* MqttException

- **createFolder**

  public int createFolder(java.lang.String path) throws MqttException

  - Description
    Makes a new folder in the SD card

  - Parameters
    * path – Path of the new folder

  - Returns – 0 if the operation was successfully completed

- **deliveryComplete**

  public void deliveryComplete (IMqttDeliveryToken token)

  - Parameters
    * token

  - See also
    * MqttCallback#deliveryComplete(IMqttDeliveryToken)

- **disconnect**

  public void disconnect () throws MqttException

  - Description
    Disconnects the client

  - Throws
    * MqttException

- **downloadFile**

  public void downloadFile(java.lang.String srcDir, java.lang.String dstDir) throws java.lang.Exception

  - Description
    Downloads a file from the SD card

  - Parameters
    * srcDir – SD card path of the file
    * dstDir – Path where the file should be store

  - Throws
    * java.lang.Exception

- **getDate**

  public java.util.Calendar getDate() throws MqttException

- Description
  Gets the current EoT device time/date

- Returns
  Calendar. The device current time/date

- Throws
  * MqttException

- **getFileSystemStructure**

public java.lang.String[] getFileSystemStructure(java.lang.String path) throws MqttException

- Description
  Gets the paths of the SD card content

- Returns
  A String[] with all the file and folder paths

- **isConnected**

public boolean isConnected ()

- Description
  Checks if the client is connected

- Returns
  true if the client is connected

- **messageArrived**

public void messageArrived(java.lang.String topic, MqttMessage messageArrived) throws java . lang . Exception

- Parameters
  * topic
  * messageArrived

- Throws
  * java.lang.Exception

- See also
  * MqttCallback#messageArrived(String, MqttMessage)

- **publish**

public void publish(java.lang.String topicName, int qos, byte [] payload) throws MqttException

- Description
  Publishes / sends a message to an MQTT server

- Parameters
  * topicName – the name of the topic to publish to
  * qos – the quality of service to deliver the message at (0,1,2) (0 in this case)
  * payload – the set of bytes to send to the MQTT server

- Throws
  * MqttException

- **removeAll**

  public int removeAll(java.lang.String path) throws MqttException

  - Description
    Removes a folder and its content recursively

  - Parameters
    * path – Path of the folder

  - Returns
    0 if the operation was successfully completed

- **removeContent**

  public int removeContent(java.lang.String path) throws MqttException

  - Description
    Removes the content of a folder (or the SD card if /mnt/sdcard is used)

  - Parameters
    * path – Path of the folder

  - Returns – 0 if the operation was successfully completed

- **removeFile**

  public int removeFile(java.lang.String path) throws MqttException

  - Description
    Removes a file from the SD card

  - Parameters
    * path – Path of the file to be removed

  - Returns

0 if the operation was successfully completed

- **resetAPConfig**

  public void resetAPConfig () throws MqttException

  - Description
    Resets the AP con.guration to the default profile

  - Throws
    * MqttException

- **setMainFrame**

  public void setMainFrame(EoT_MainFrame frame)

  - Description
    Sets the main frame where results are displayed

  - Parameters
    * frame

- **subscribe**

public void subscribe(java.lang.String topicName, int qos) throws MqttException

  - Description
    Subscribes the client to a topic on an MQTT server

  - Parameters
    * topicName – to subscribe to (can be wild carded)
    * qos – the maximum quality of service to receive messages at for this subscription

  - Throws
    * MqttException

- **unsubscribe**

public void unsubscribe(java.lang.String topicName) throws MqttException

  - Description
    Unsubscribes the client from a topic

  - Parameters
    * topicName

  - Throws
    * MqttException

- **updateDate**

public int updateDate(java.lang.String year, java.lang.String month, java.lang.String day, java.lang.String hour, java.lang.String mins, java.lang.String secs) throws MqttException

  - Description
    Changes the EoT device time/date

  - Parameters
    * year
    * month
    * day
    * hour
    * mins
    * secs

  - Throws
    * MqttException

- **uploadFile**

public int uploadFile(java.lang.String srcDir, java.lang.String dstName) throws java.lang.Exception

  - Description
    Sends a file to the SD card

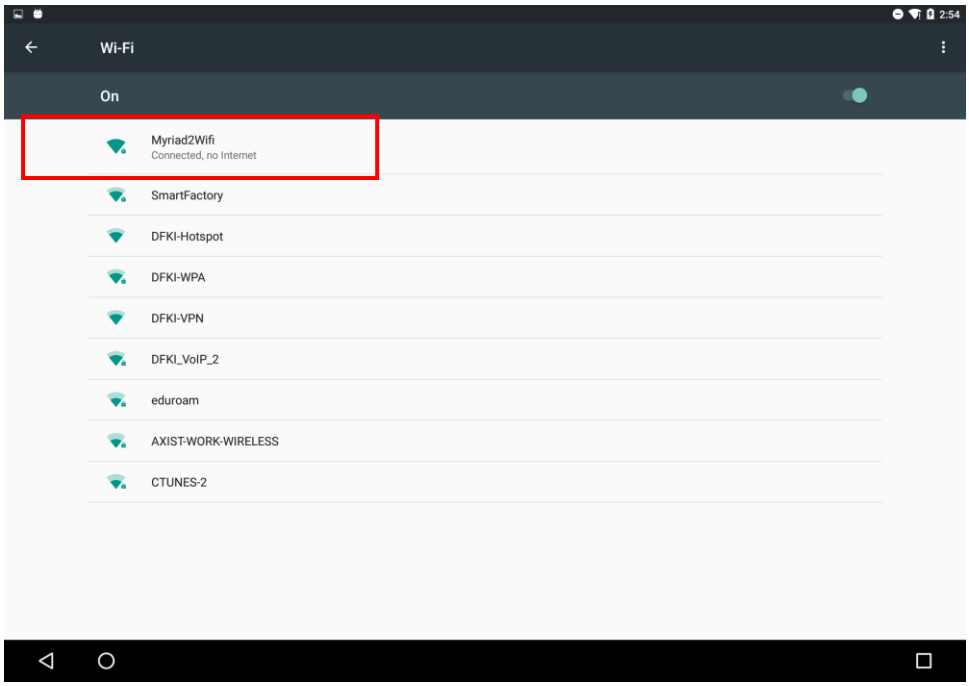
  - Parameters
    * srcDir – Path of the file
    * dstName – SD card path where the file should be store

  - Returns
    0 if all is OK

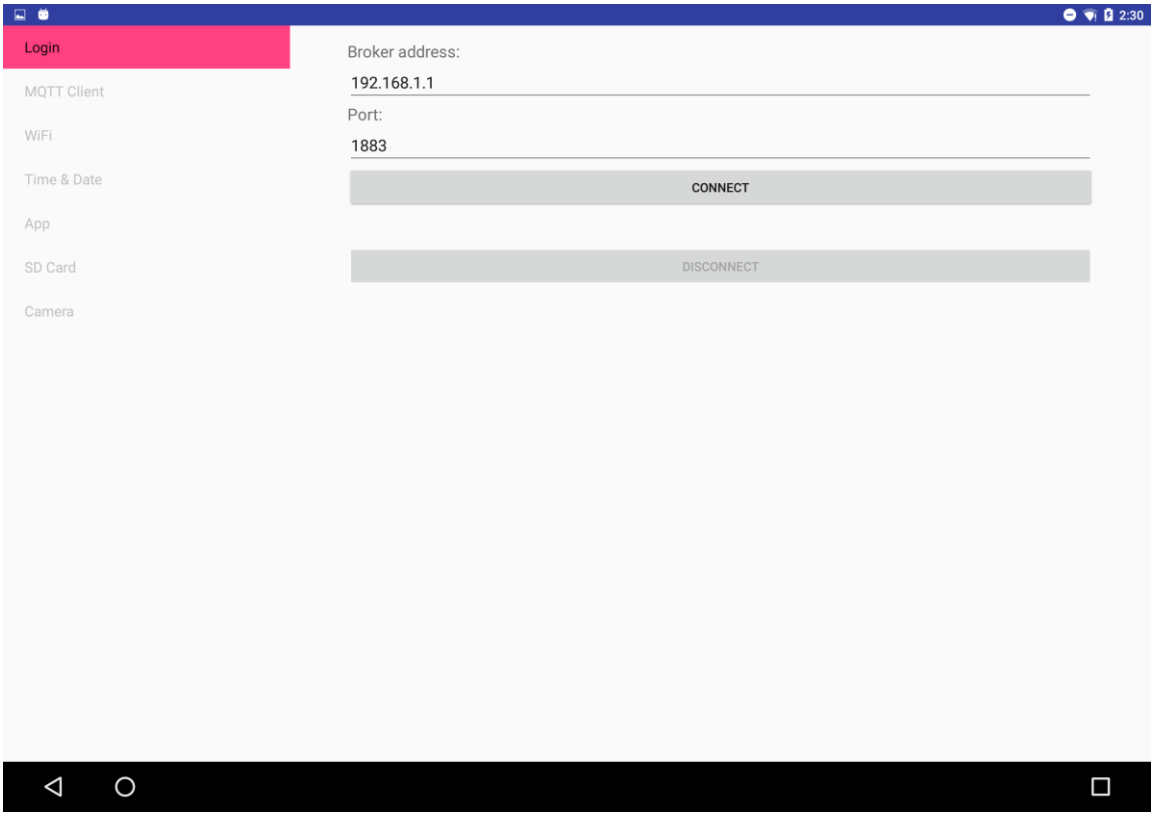
  - Throws
    * java.lang.Exception

6.3 User Interface/Use of the Application

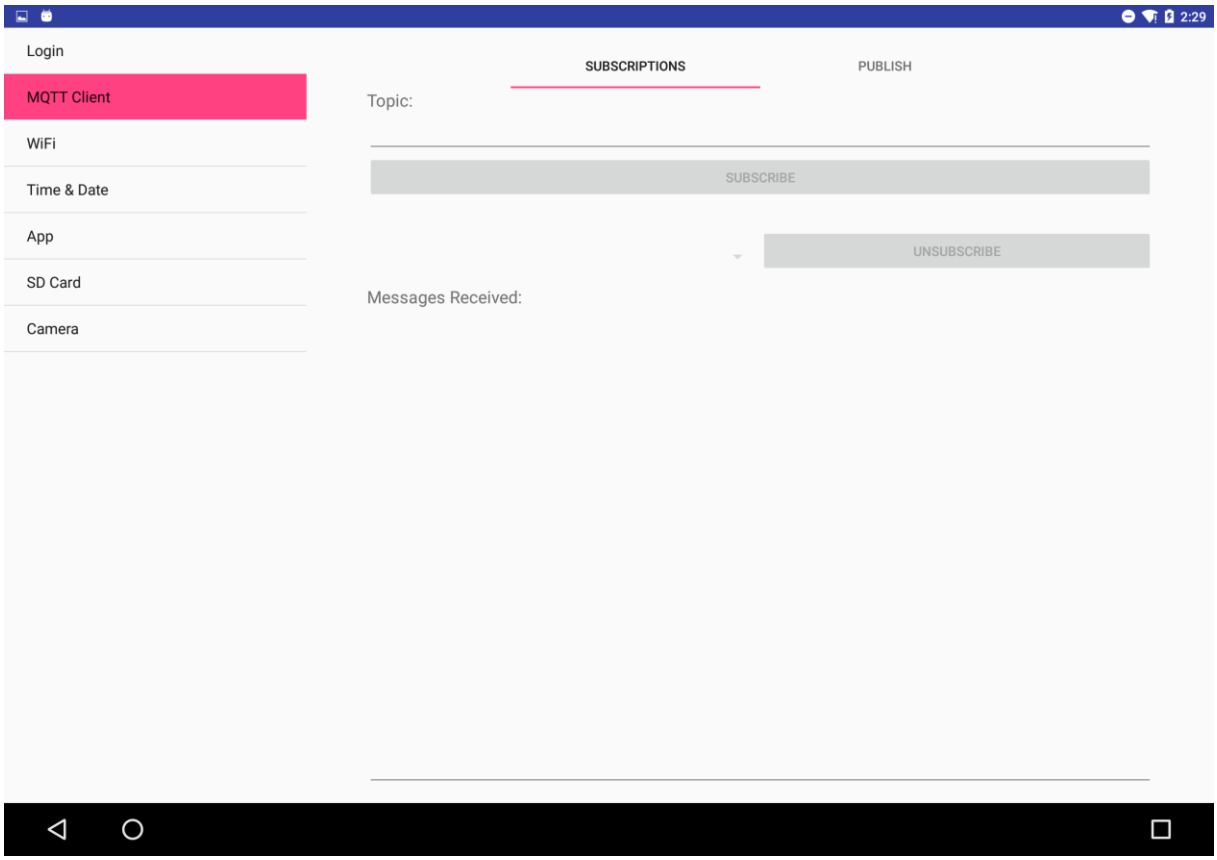
The application is divided into seven panels: Login, MQTT Client, WiFi, Time & Date, App, SD Card and Camera.
Before connecting the EoT Control Mode Android application to the EoT device the mobile device should be connected to the EoT device AP.

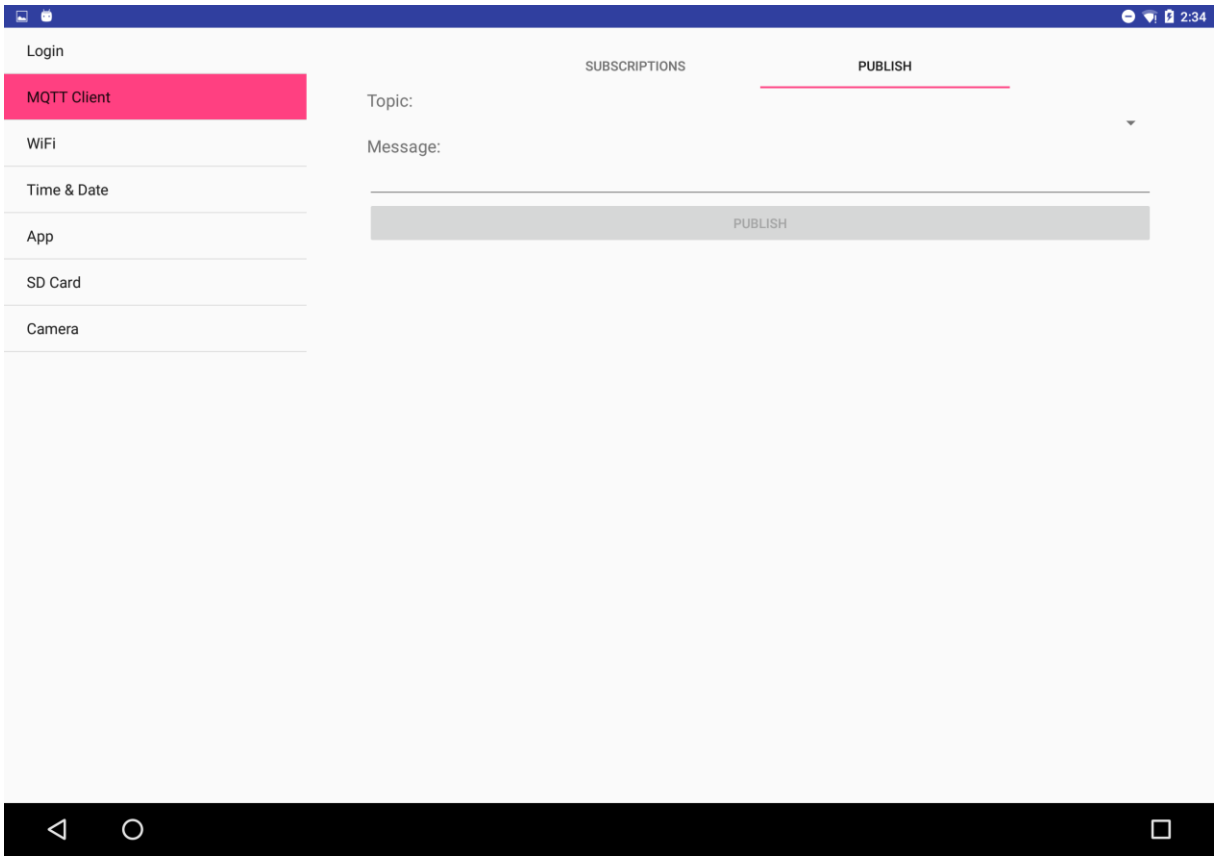


Once the mobile device is connected to the EoT device AP, the MQTT client can be connected to the Pulga broker using the correct IP address and port.



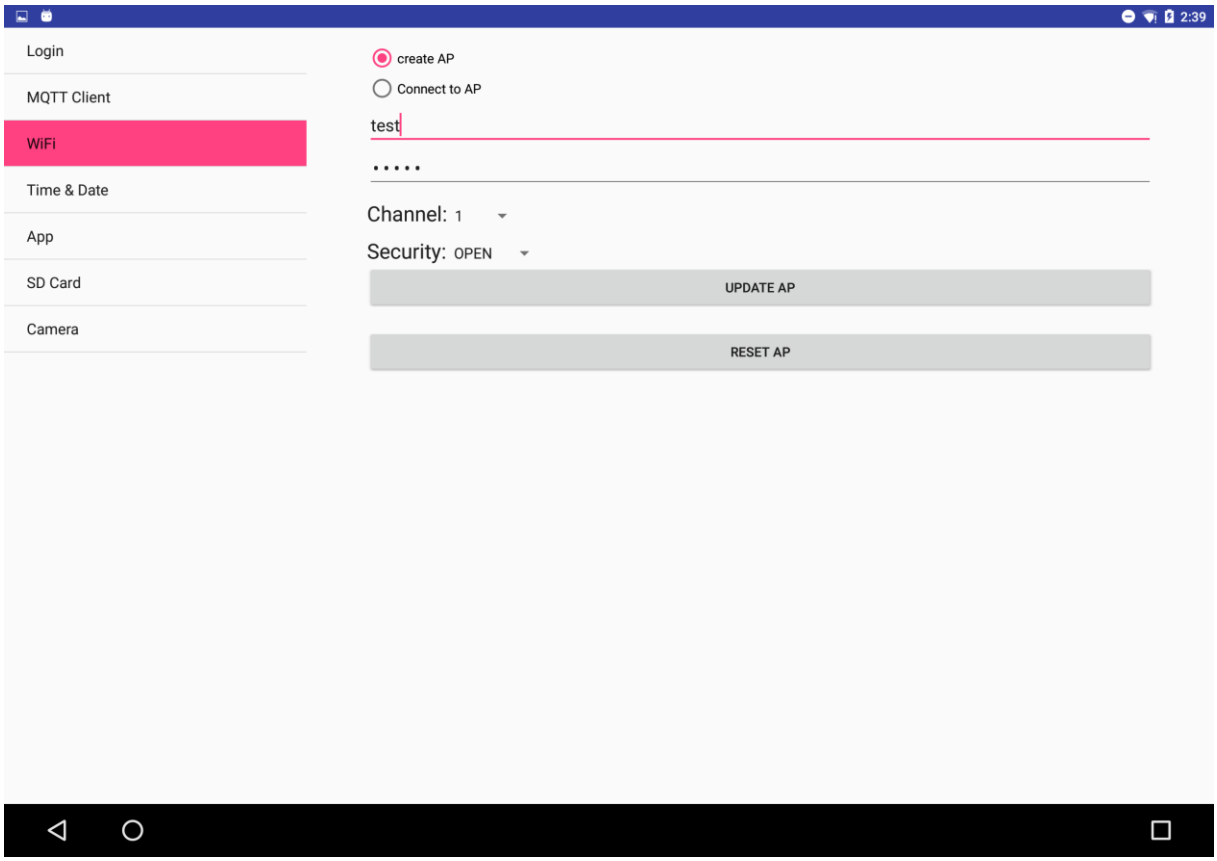
After that, it is possible to use the application as a common MQTT client, performing topic subscriptions and publishing messages to topics.





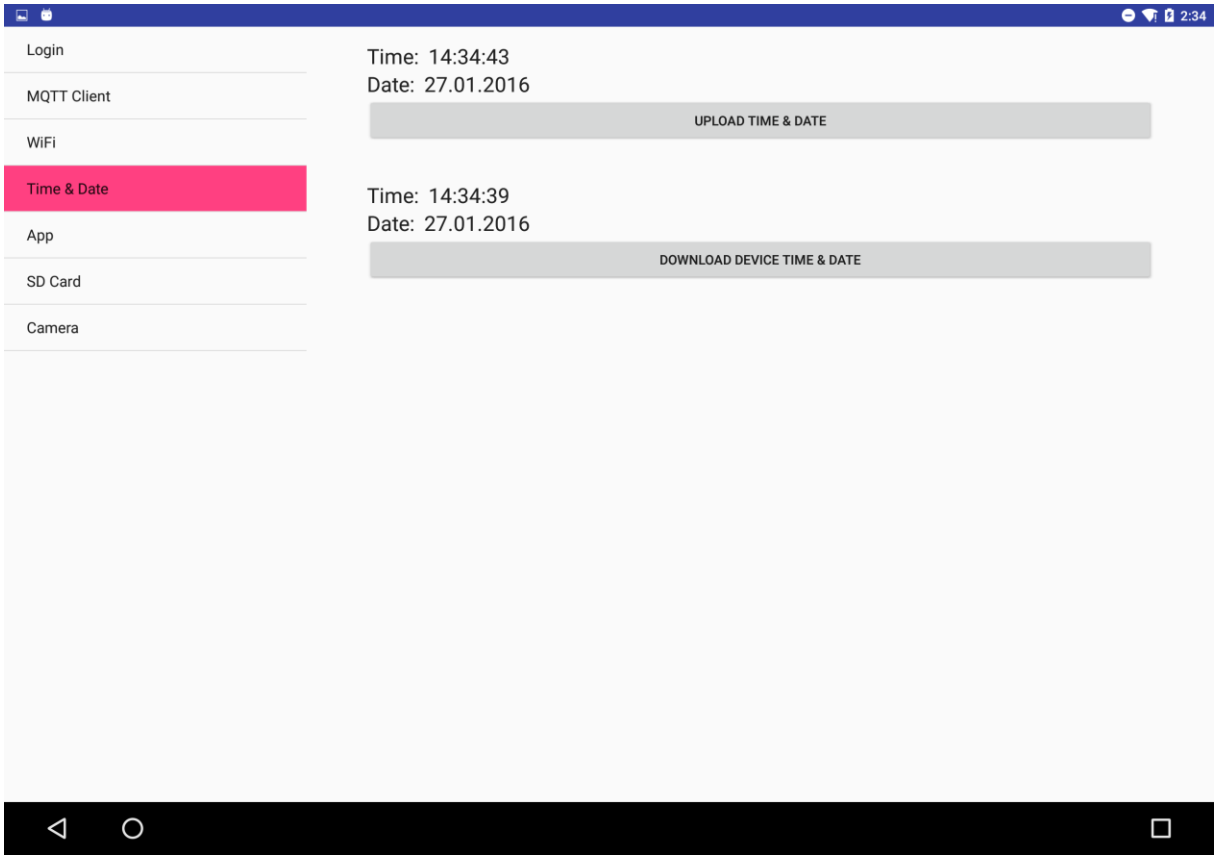
In the WiFi panel there are options that allow the user to configure WiFi. The EoT device WiFi configuration includes options to:

- create an AP with new parameters,
- connect the device with an existing AP and
- reset the device AP settings to the default profile.

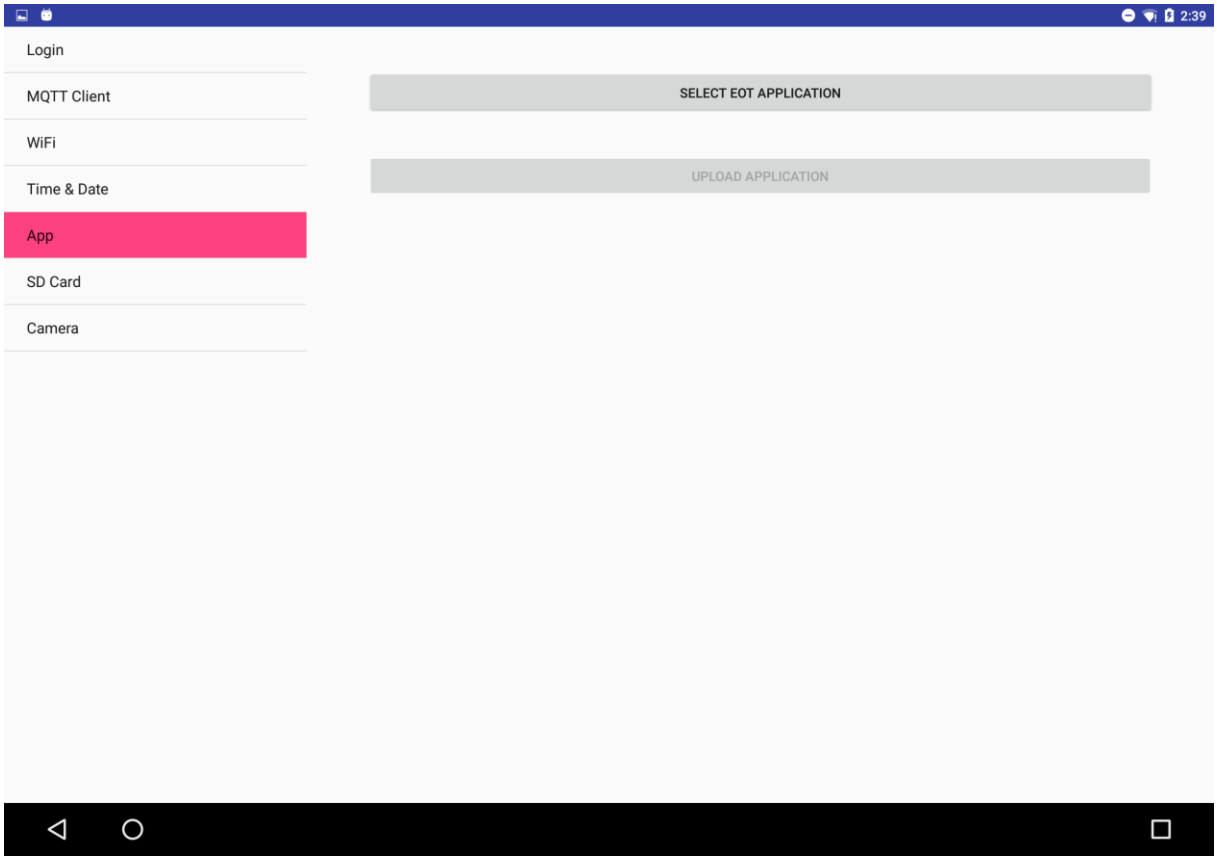


If the device's WiFi configuration is changed, the user needs to connect the mobile device to the new AP or the same wireless network the EoT device is connected to. Then, the client-broker connection is re-established.

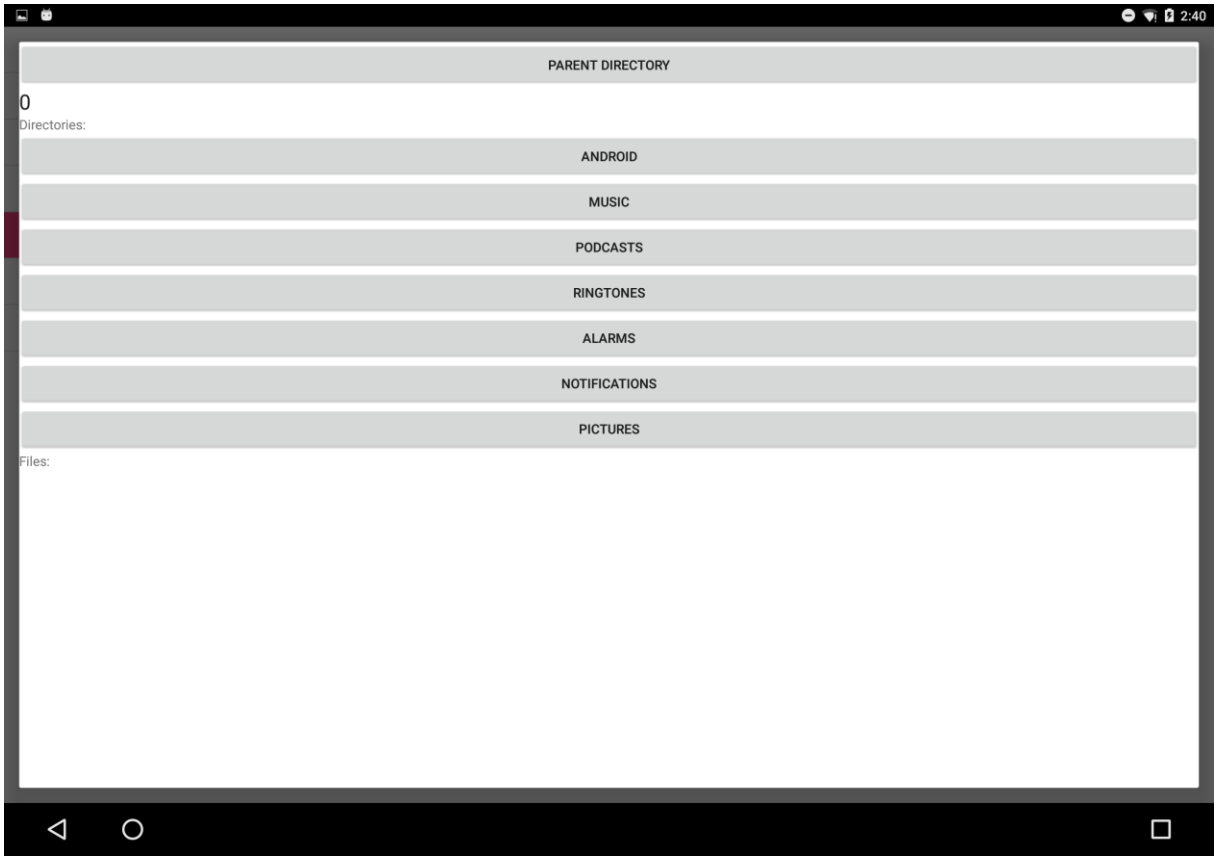
In order to set the current time values in the EoT device it is possible to use the Date & Time settings provided in the Android application. This allows the user to get the current time and date of the mobile device and set them in the EoT device. In addition, it is possible to check the current time of the EoT device.



In the App panel the user can upload an EoT application to the EoT device.

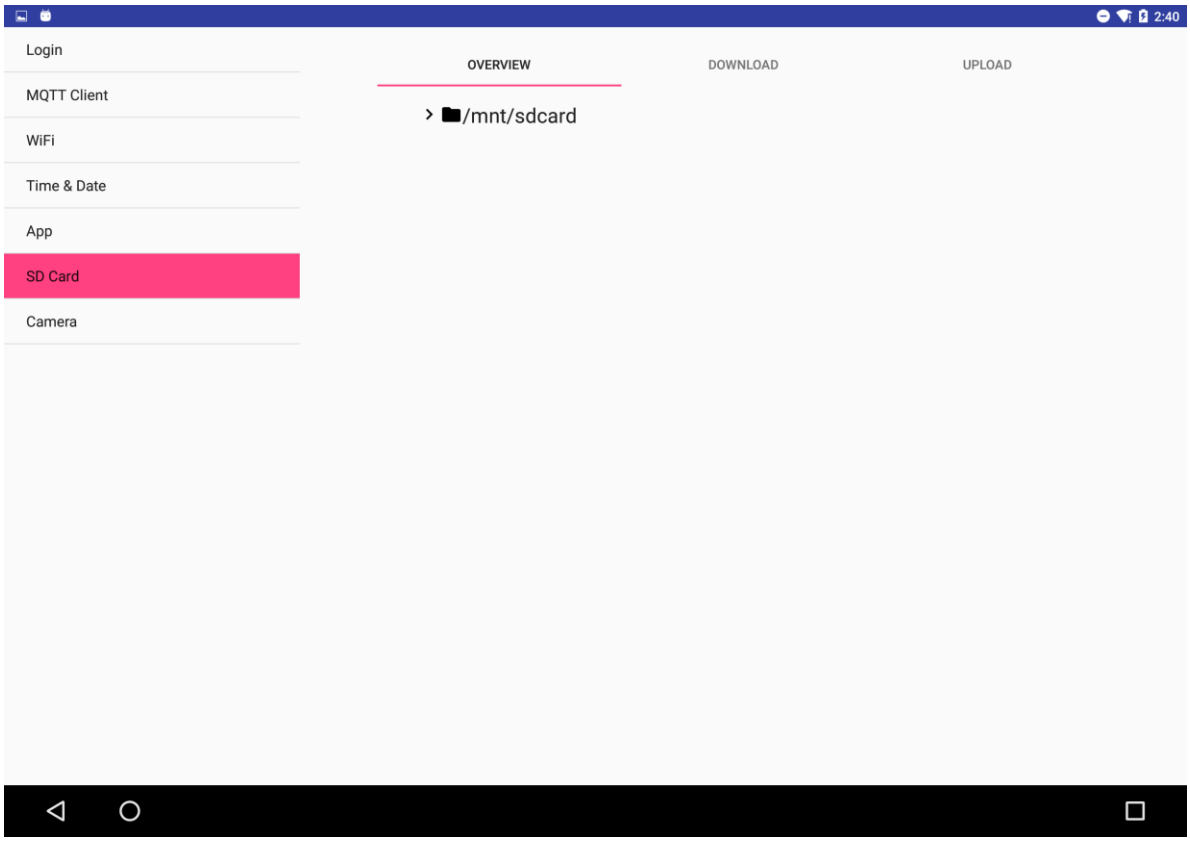


The process consists of two steps. First the user selects an EoT application.

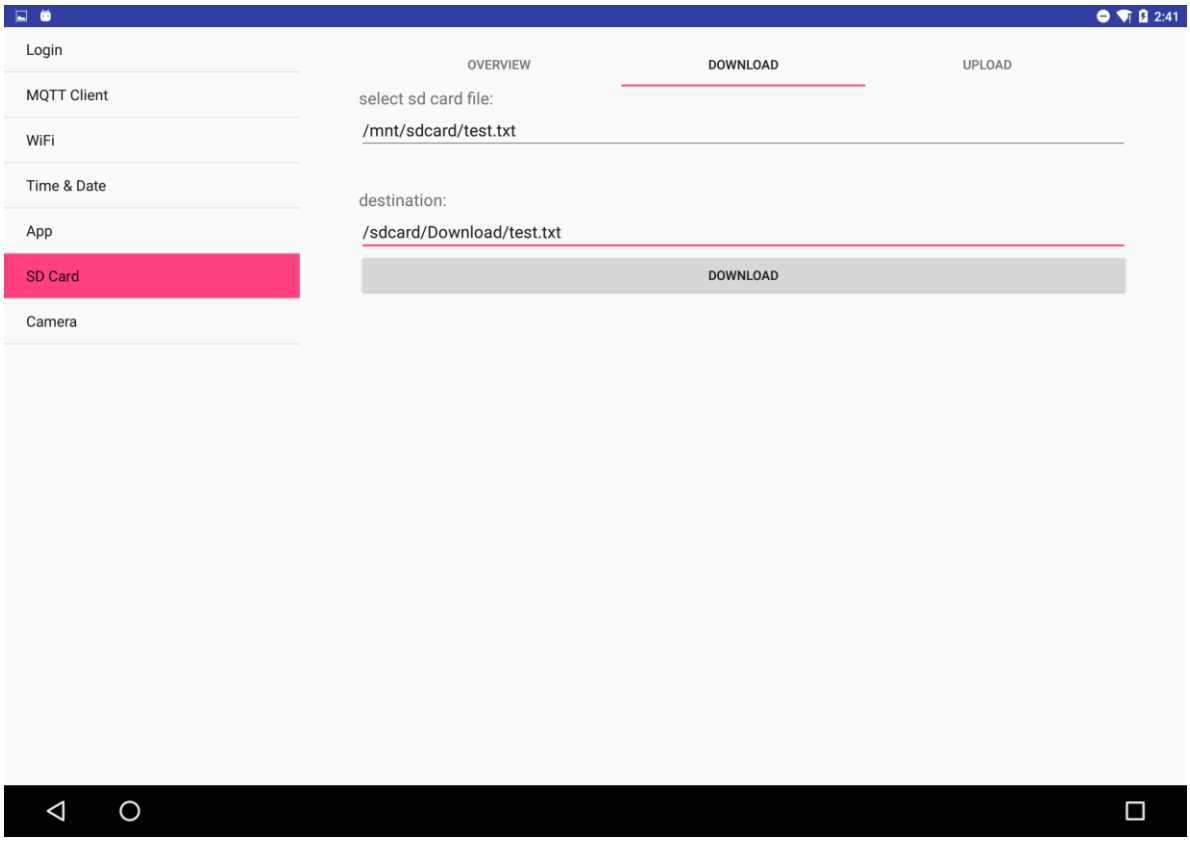


Then the user hits the “upload application” button to initiate the upload process.

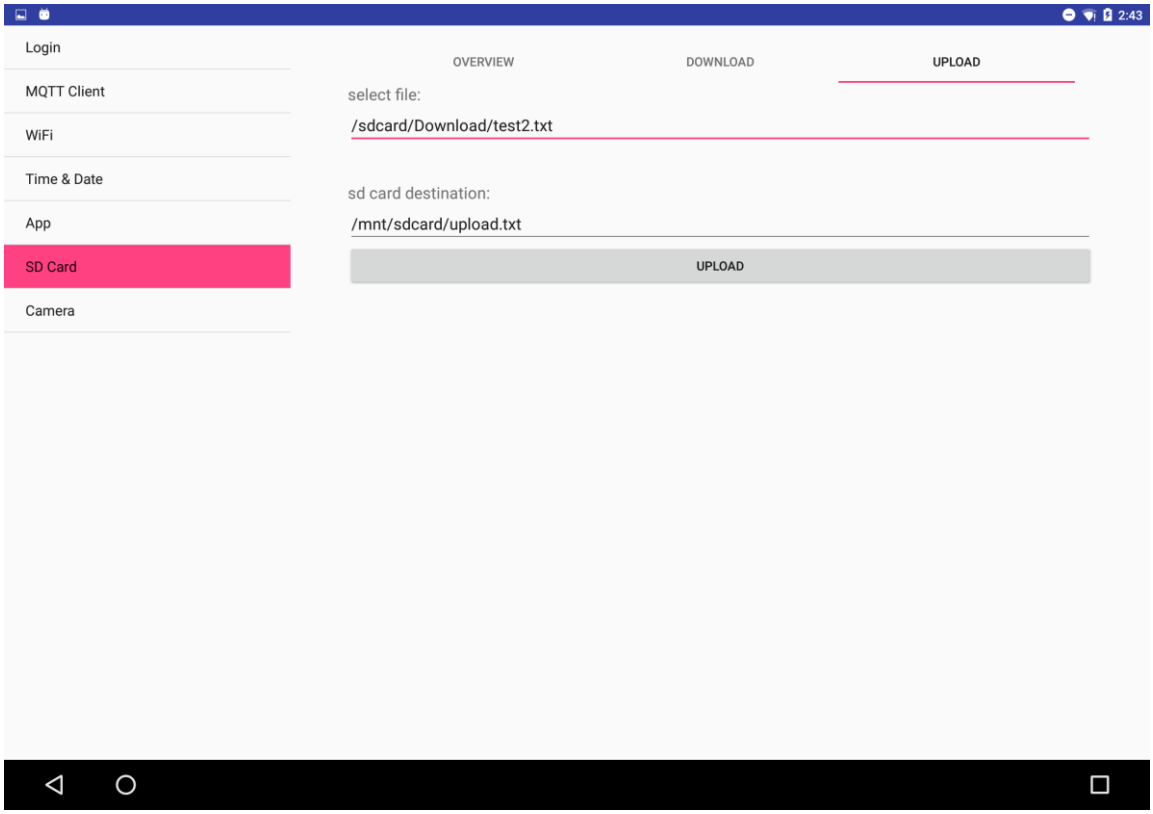
Finally, the SD card management options are divided into three parts. The first part shows the directory tree of the SD card.



Files can be downloaded from SD card to the mobile device in the second tab.

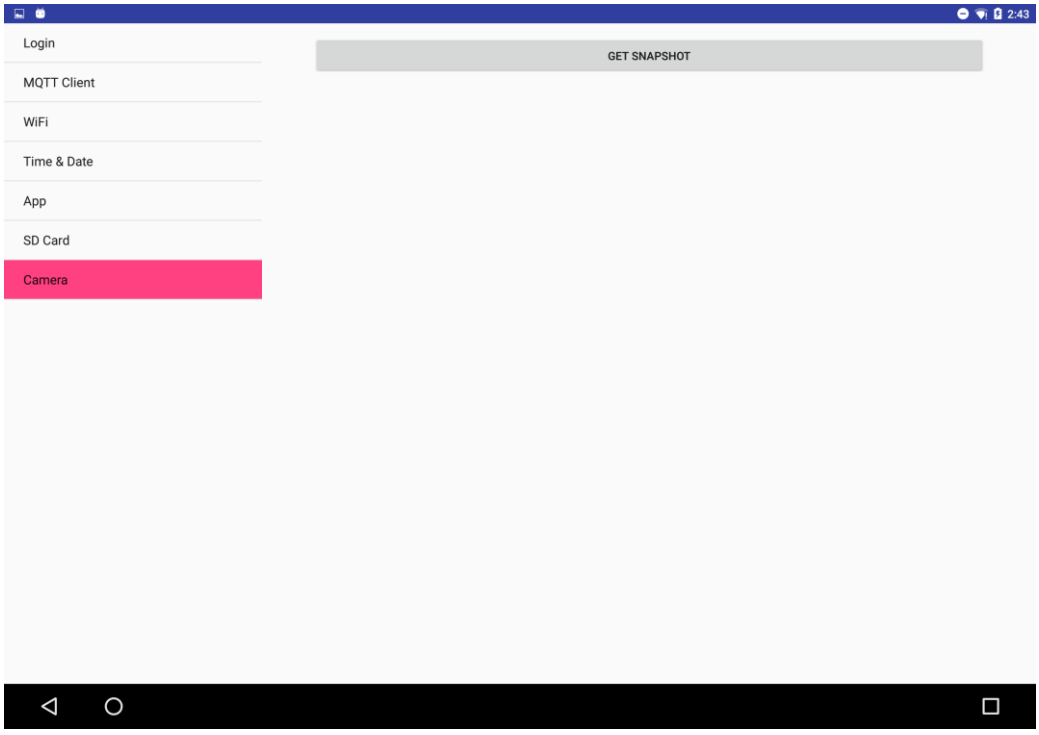


In the third tab the user can upload files to the SD card of the EoT device.



Note that when the EoT device is in AP mode (by default) only one client can be connected to it. This is considered a desirable feature in terms of security.

In the camera panel the user can request a snapshot from the EoT device's camera.



6.4 Open Issues

Two functionalities are implemented in the described EoT configuration and control app for Android but not in Pulga. In particular:

- The snapshot retrieval needs to be implemented when the new camera becomes ready.
- The functionality related to the flash needs to be implemented when the conflicts between the WiFi chip and the flash memory are resolved in the new version of the hardware.

7 CODE

The code of the EoT project can be found in the following GitLab repository:

https://gitlab.com/espiaran/EoT

The Pulga code can be found in the myriad applications directory of the WP3:

WorkPackage_3/myriad/apps/pulga_control_app

Pulga depends on *Crypto, SDCardIO, WifiFunctions,* and *TimeFunctions* modules. These modules can be found under the *WorkPackage_3/myriad/libs* folder.

The Java Control Mode Android application is stored in the following directory:

WorkPackage_3/mobile/android/apps/EoT_control_app

This app needs the Paho library which is downloaded automatically by gradle when building the app in Android Studio.

Please note that the Android App requires Android version 5 (Lollipop) or later.

8 CONCLUSION

EoT focuses on developing an open platform for mobile embedded computer vision. The building elements have been all optimized for size and cost. Particularly, the device optimizes the processing power vs energy consumption ratio. Apart from hardware and architectural elements, software and protocols used have been optimized as well. The publish/subscribe MQTT protocol has been selected early on because of its low-power profile. While typical scenarios involve (mobile) clients sending and receiving messages to/from a cloud-based broker, a novel architecture is proposed in which each EoT device can act as a broker itself. This provides a minimal way of communication that does not require any cloud-based broker. This way no data is initially sent through the Internet which is also an advantage in terms of security. This basic form of communication can be in turn used to establish additional modes of communication should the application require it. It can, for example, be used to configure the device and the embedded application to run on it. This includes setting up a connection to an existing WiFi network.

The proposed embedded MQTT broker, Pulga, provides the functionality to install and configure applications in the EoT device using a desktop computer or a mobile device with any MQTT client. It includes all main functionalities of a classic MQTT broker as well as the new functionality required for EoT. This new functionality is available through Pulga's Java API. The implementation of the client side of this API for Android devices as well as the usage of the Android app has been described in this document.

9 REFERENCES

[1]. Banks, A., & Gupta, R. (2014). MQTT Version 3.1. 1. OASIS Standard. https://www.oasis-open.org/standards. Last accessed: 12th of January 2016.
[2]. MQTT: a machine-to-machine (M2M)/Internet of Things connectivity protocol. http://mqtt.org. Last accessed: 12th of January 2016.
[3]. http://eclipse.org/paho . Last accessed: 12th of January 2016.

10 GLOSSARY

EoT	Eyes of Things
SoC	System on a Chip
HTTP	Hypertext Transfer Protocol
MQTT	Message Query Telemetry Transport
API	Application Programming Interface
IoT	Internet of Things
TCP/IP	Transmission Control Protocol / Internet Protocol
PC	Personal Computer
SSID	Service Set Identifier
OASIS	Organization for the Advancement of Structured Information Standards
M2M	Machine to Machine
QoS	Quality of Service
SD	Secure Digital
AP	Access Point
SSL/TLS	Secure Sockets Layer / Transport Layer Security

- End of document -