



UNIVERSITÀ  
degli STUDI  
di CATANIA

*Department of Math and Computer Science  
Bachelor's degree in Computer Science*

PARK SMART

# **Development of a multi-threaded embedded software for remote management using C++ and MQTT**

*Salvatore Campisi*

30 November 2018 – AY 2017/2018

Supervisors:

*Prof. Eng. Salvatore Antonio Riccobene*

*Dr. Giuseppe Patanè*



## Park Smart

Park Smart is a Sicilian solution company which operates in the field of Smart Cities.

The company's goal is **monitoring** the parking areas of the city and **communicate** the parking **availability** to city's inhabitants.



How?

# Artificial Intelligence

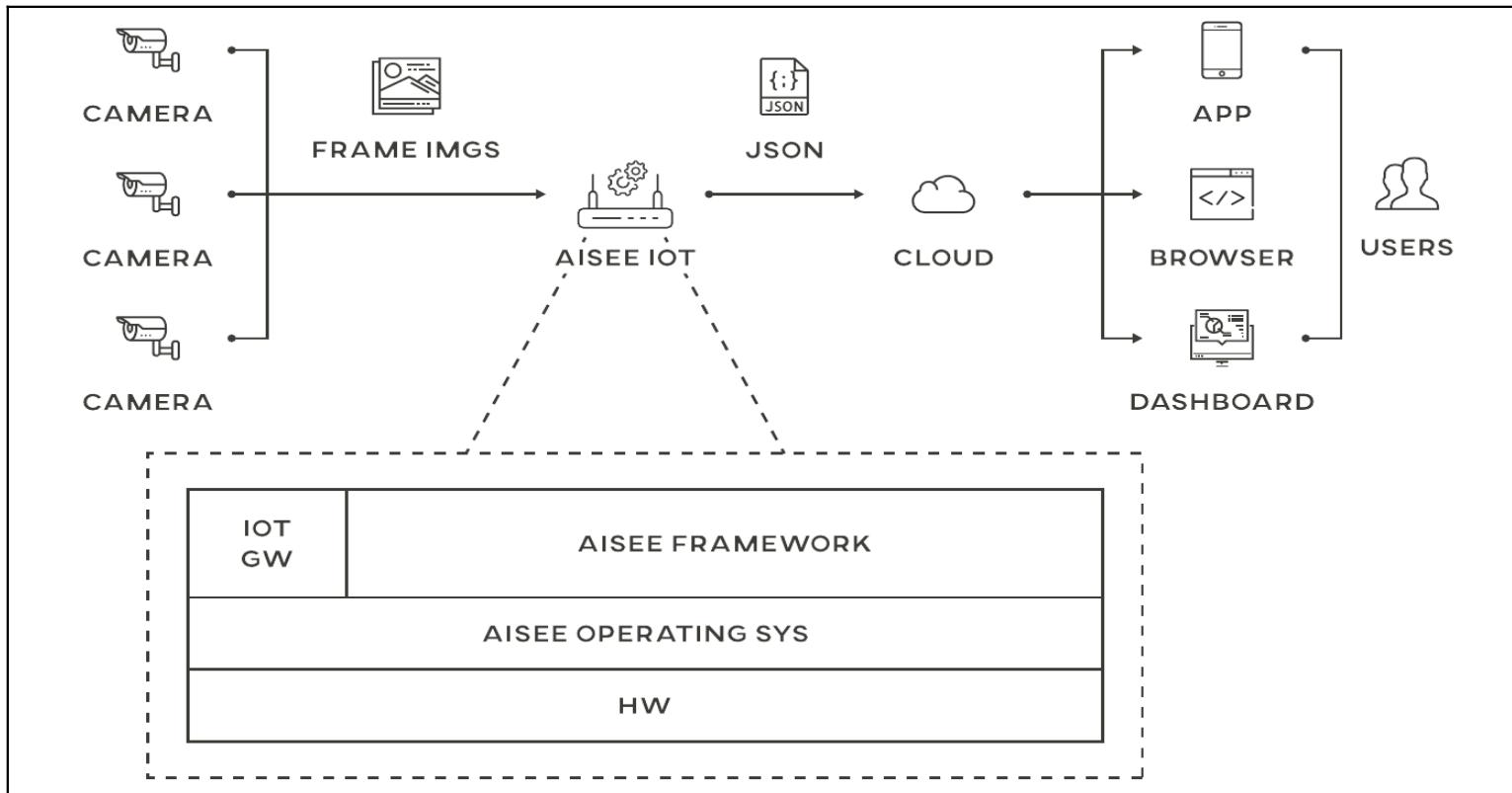
# Computer Vision

# Cloud System

# Mobile Application



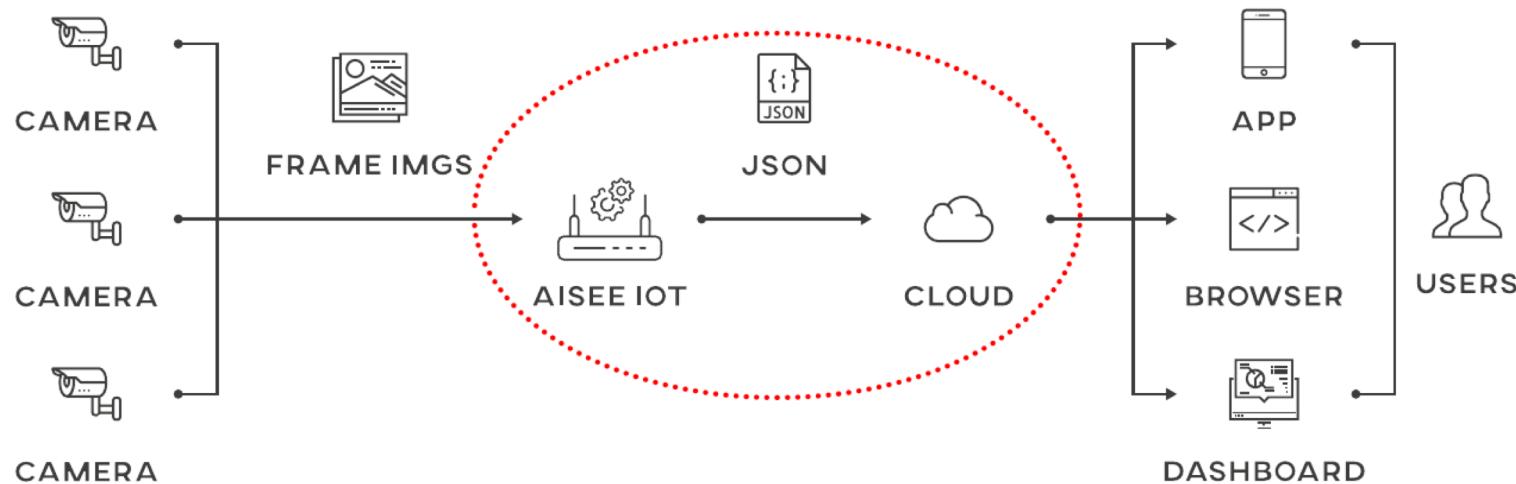
## Park Smart Infrastructure





## Final Project Main Goal

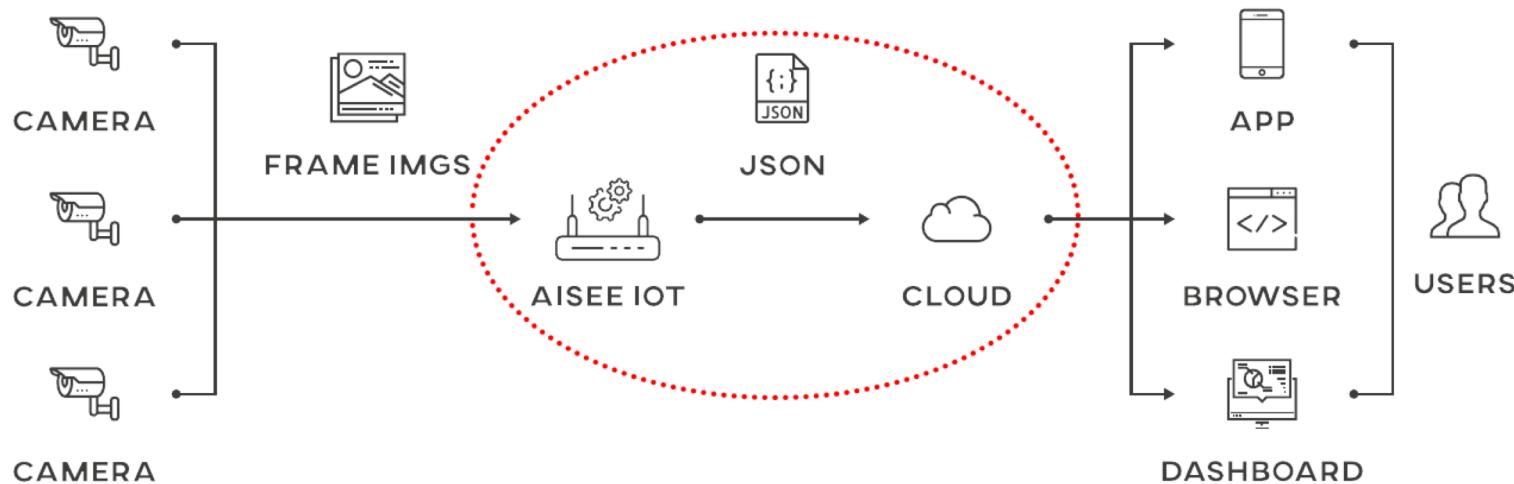
Develop a **software** for allowing the *AISee Box* device (Computer Vision system) to communicate **efficiently** and **asynchronously** with the company's Cloud System.





## Other Goals

1. **Stable**
2. **Modular**
3. **Multithread**
4. **Cross Platform**





UNIVERSITÀ  
degli STUDI  
di CATANIA

*Department of Math and Computer Science  
Bachelor's degree in Computer Science*

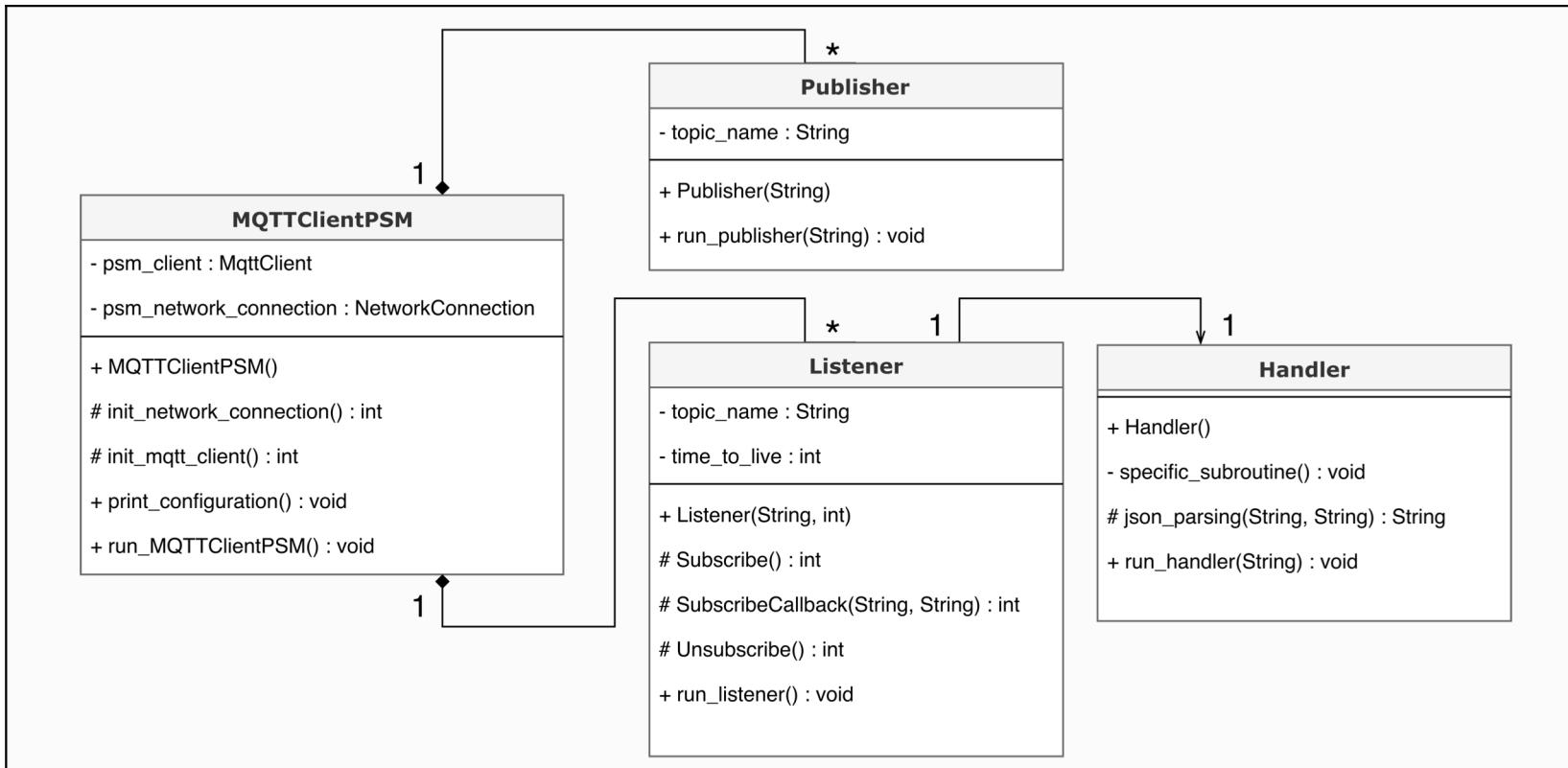
PARK SMART

## Used Technologies





## Software Desing (a part of)





# Software Development (a part of)

```
int Listener::Subscribe(){
    cout << "[Listener][INFO] I'm doing the Subscribe on the topic: " << topic_name << endl;

    // Linking the ApplicationCallbackHandlerPtr to SubscribeCallback
    awsiotsdk::mqtt::Subscription::ApplicationCallbackHandlerPtr p_subscribe =
        bind(&Listener::SubscribeCallback, this, placeholders::_1, placeholders::_2);

    // Using the SDK API for making the subscribe operation:
    shared_ptr<awsiotsdk::mqtt::Subscription> p_subscription =
        awsiotsdk::mqtt::Subscription::Create(move(topic_name), QOS0, p_subscribe, nullptr);

    awsiotsdk::util::Vector<shared_ptr<awsiotsdk::mqtt::Subscription>> topic_vector;
    topic_vector.push_back(p_subscription);

    int rc = temp_mqtt_client->Subscribe(topic_vector, awsiotsdk::ConfigCommon::mqtt_command_timeout);

    // Checking if subscription is OK:
    if(rc != 0) cout << "[Listener][FAILURE] Impossible make the Subscribe." << endl;
    else cout << "[Listener][OK] Subscribe correctly made up." << endl;

    return rc;
}
```



# Software Development (a part of)

```
int Listener::SubscribeCallback(String topic_name, String payload){  
    cout << "-----" << endl;  
    cout << "[Listener][INFO] Message on topic: [" << topic_name << "]" << endl;  
    cout << "[Listener][INFO] Message length: " << payload.length() << endl;  
    cout << "[Listener][INFO] Message content:" << payload << endl;  
    cout << "-----" << endl;  
  
    // Starting a new thread:  
    thread th_handler(&Handler::run_handler, payload);  
    // This thread will continue the execution and th_handler executes independently:  
    th_handler.detach();  
  
    return 0;  
}
```



# Software Development (a part of)

```
int run_publisher(String temp_message){
    cout << "===== START PUBLISHING ON A TOPIC =====" << endl;
    cout << "[Publisher][INFO] Publishing on topic: [" << topic_name << "]"
        << endl;
    cout << "[Publisher][INFO] Message length: "
        << temp_message.length() << endl;
    cout << "[Publisher][INFO] Message content:" << temp_message << endl;

    // Using the SDK API for making the publish operation:
    int rc = temp_mqtt_client->PublishAsync(topic_name, false, false, QOS1,
                                              temp_message, nullptr, 0);

    // Checking if publish operation went OK:
    if(rc == 0) cout << "[Publisher][OK] Message correctly published on topic."
        << endl;
    else cout << "[Publisher][FAILURE] Message not published on topic."
        << endl;

    return rc;
}
```



# Software Execution and Testing

## Start and Inizialization

```
===== STARTING PARKSMART MQTT CLIENT =====

===== CREATING PSM_MQTT_CLIENT =====
[MQTTClientPSM] [OK] MbedTLS connection correctly established.
[MQTTClientPSM] [OK] Correctly created psm_mqtt_client
[MQTTClientPSM] [OK] Connection to endpoint accepted.
[Handler] [INFO] Handler correctly built.
[Publisher] [INFO] Publisher correctly built.
[Listener] [INFO] Listener correctly built.
[MQTTClientPSM] [INFO] MQTTClientPSM correctly built.
=====

Endpoint: a3rzums6u8jbaw-ats.iot.eu-west-1.amazonaws.com
MQTT port: 8883
Root CA Certificate: /certs/root-CA-1.crt
Device Certificate: /certs/eb9a395f1c129dc427a8c0060307a7bd70f571fa1c5d799b4523bee7e8aa0f66.certificate.pem.crt
Device Private Key: /certs/eb9a395f1c129dc427a8c0060307a7bd70f571fa1c5d799b4523bee7e8aa0f66.private.pem.key
Client ID: eb9a395f1c129dc427a8c0060307a7bd70f571fa1c5d799b4523bee7e8aa0f66
Classification Config Path: /data/classification_tool/config
Img Path: /data/classification_tool/img_dataset
Model Path: /data/classification_tool/models
Firm Path: /data/fw
Class Path: /data/classification_tool/res_class
Http Config Path: /data/classification_tool/res_class
```



# Software Execution and Testing

## Publishing on a MQTT topic

```
===== START PUBLISHING ON A TOPIC =====
[Publisher] [INFO] Publishing on topic: [boards/eb9a395f1c129dc427a8c0060307a7bd70f571fa1c5d799b4523bee7e8aa0f66/status]
[Publisher] [INFO] Message length: 40
[Publisher] [INFO] Message content: STATUS UPDATE from Parksmart MQTT Client
[Publisher] [OK] Message correctly published on topic.
=====
```

A screenshot of the AWS CloudWatch Logs interface. The top navigation bar shows the AWS logo, 'Services' dropdown, and 'Support' dropdown. The main area displays a log entry with the following details:

boards/eb9a395f1c129dc427a8c0060307a7bd...08 nov 2018 16:55:35

Esporta Nascondi

STATUS UPDATE from Parksmart MQTT Client

The log entry corresponds to the message published in the terminal output above.



# Software Execution and Testing

Listening on a MQTT topic and handling of the received command

The screenshot shows the AWS Lambda function code editor. The URL in the browser bar is `boards/eb9a395f1c129dc427a8c0060307a7bd70f571fa1c5d799b4523bee7e8aa0f66/`. The code in the editor is:

```
1  {
2    "psm_command": "update_your_status"
3 }
```

The terminal output shows the execution of the Lambda function, starting with the message "START LISTENING ON A TOPIC". It then logs the subscription to the specified topic and receives a message containing the JSON object defined in the code. Finally, it logs the execution of the subroutine corresponding to the received command.

```
===== START LISTENING ON A TOPIC =====
[Listener] [INFO] I'm doing the Subscribe on the topic: boards/eb9a395f1c129dc427a8c0060307a7bd70f571fa1c5d799b4523bee7e8aa0f66/
[Listener] [OK] Subscribe correctly made up.

[Listener] [INFO] Message on topic: [boards/eb9a395f1c129dc427a8c0060307a7bd70f571fa1c5d799b4523bee7e8aa0f66/]
[Listener] [INFO] Message length: 40
[Listener] [INFO] Message content:
{
  "psm_command": "update_your_status"
}

[Handler] [OK] Received Command: update_your_status
Executing the Subroutine ...
... Subroutine correctly executed
```