

Pelion End-to-End application

Final Report

Version 1.0

Ville Karsikko 2463933
Bekim Abazi 2425528
Tomi Lehto 2508005
Atte Jauhiainen 2433947
Markus Savusalo 2434205

1.Executive abstract

We are currently in the beginning of the era of internet of things, which is why established tools for device management are still lacking. In this document we design, implement and test two semi-independent projects. One part is developed for the internet of things device, and other is a data and device management dashboard web application. The main goal of this project is to demonstrate the end-to-end capabilities of existing infrastructure.

521479S Software project (2018)

Pelion End-to-End application

Contents

Executive abstract	2
Introduction	4
Customer	4
Acronyms and Definitions	5
References	5
Project Description	5
Deployment	5
Implementation	5
Issues	9
Realised requirements	10
Used software development method	10
Testing / acceptance of the software	10
Project timeline	13
Feedback to the course organizers	13

2.Introduction

The project consists of two main components. The first part is programming an embedded device running ARM Mbed OS [1] which will connect to and handle data from Bluetooth low energy -beacon devices. The embedded device is then connected to existing infrastructure - a platform for managing IoT devices called Pelion [2]. The other half of the project is a web application, which utilizes the REST interface provided by Pelion. The goal of the web application is to implement extended functionality to Pelion. This functionality could be, for example, plotting devices on a map based on their GPS-coordinates and presenting data collected from beacon devices.

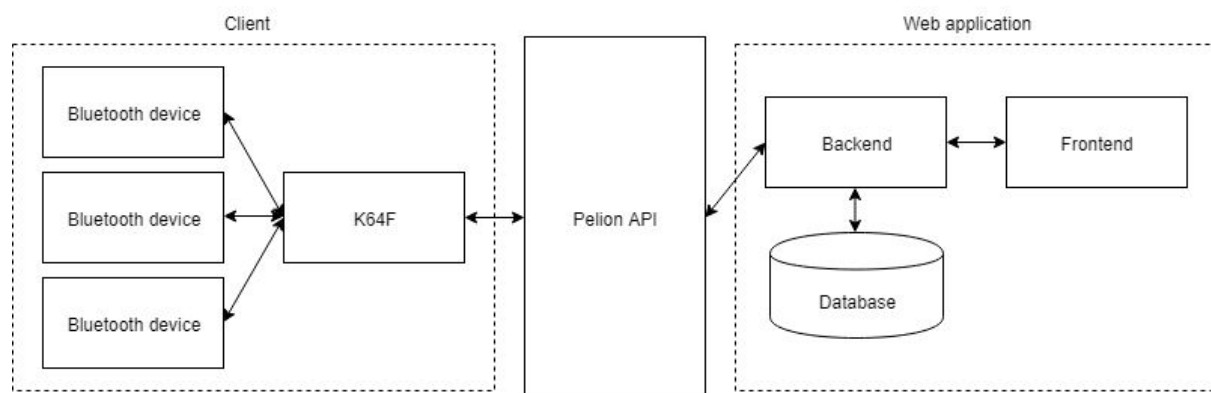


Figure 1. High level architecture of this project.

In figure 1 one can see the higher level architecture of this project. The two components, web application and the IoT devices, are connected together via Pelion. Pelion is a device management interface, made specifically by ARM. Pelion allows keeping track of devices and their attributes and querying this data through a REST interface. It acts as a message broker between the IoT devices and other applications.

3.Customer

Our customer is Arm Finland Oy. We are working with the ARM Mbed IoT devices and we utilize the existing Pelion cloud service, which enables connectivity between the devices and an arbitrary web service. Our contact person from ARM was Jukka Kansanaho.

4. Acronyms and Definitions

IoT	Internet of Things
REST	Representational State Transfer
URL	Uniform Resource Locator
BLE	Bluetooth low energy
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver Transmitter

5. References

- [1] ARM Mbed OS documentation, <https://os.mbed.com/>
- [2] ARM Pelion IoT Platform, <https://www.arm.com/products/iot/pelion-iot-platform>
- [3] BLE Board HW patch https://os.mbed.com/teams/ST/code/X_NUCLEO_IDB0XA1/
- [4] BLE Board FW update <https://os.mbed.com/teams/ST/code/BlueNRG-MS-Stack-Updater/>
- [5] Mbed OS unit testing, <https://os.mbed.com/docs/v5.10/tools/unit-testing.html>
- [a] <https://github.com/tomlehto/Pelion-E2E-Client>
- [b] <https://github.com/Kumikurre/Pelion-E2E-WebApp>

6. Project Description

a. Deployment

The development deployment is on the localhost. However, even then the web services need to be exposed to the internet. This is necessary to enable the callback URL functionality. Without having the RESTful service published it would be impossible to receive data from Pelion cloud asynchronously. Pelion cloud pushes data to a set URL, which needs to be accessible. In the development deployment the backend and database are deployed using Docker. Frontend is served with the Node live server. In production, both the backend and frontend are served with Nginx web server.

The client side software was developed for an NXP FRDM K64F development board and compatible Wi-Fi (ESP8266-based Grove UART-WiFi adapter) and BLE (ST X-NUCLEO-IDB05A1) extension boards.

b. Implementation

Client-side implementation for the K64F board consisted mainly from two parts, which were connectivity and Pelion resource management.

521479S Software project (2018)

Pelion End-to-End application

One of the first things we implemented in this project was Wi-Fi connectivity, which made everything else easier since an ethernet connection was not needed every time a new change had to be tested. The only downside to the Wi-Fi connectivity was that it was not possible for us to change the WLAN we were connected to or switch to ethernet during runtime. Every time we wanted to change either, the project had to be built again with different options.

The other important connectivity feature on the client side was BLE. The BLE extension board connected to our main board through SPI with BlueNRG-Cordio-driver. MbedOS platform provides a BLE-library with a plenty of features for BLE connectivity. The library is object-oriented where the BLE-board is treated as an object, with methods for different functionalities. Our implementation defines a scanning method for the class, which we call in our code to do a BLE scan. The scan updates the temperature values to client memory, and after that the values are updated to Pelion Cloud with Pelion API.

A lot of effort was put into managing the Pelion resources and simulating beacon device data. We ended up with a implementation that allows a static number of resources to be registered to Pelion and the BLE devices will be paired up with these resources. The validity of each resource/device is indicated to Pelion with a simple bitmap. Originally we planned to have a dynamic number of resources that would be registered to Pelion on the fly but we found out that the Pelion platform didn't support that kind of functionality since the resources had to be created before registering the device to Pelion. Any changes to the resource parameters (other than the values) after registration simply didn't do anything. This is probably due to security reasons especially for industrial use of the Pelion platform. Simulations were done so that the K64F enabled 1-10 resources corresponding to BLE devices and generated temperature sensor values for them. This data could then be used in developing the web app.

The web application implementation consists of two parts, as seen in figure 1. It consists of backend and frontend. Also, there is a MongoDB database for storing data. Backend is programmed in Python and utilizes the Flask-library to implement the REST api and its endpoints. The frontend is programmed in Angular. Backend requires a connection to Pelion to function, as the Backend does not hold track of the state of the devices or their resources. This data is queried from Pelion on demand. Pelion, however, does not store the data that the IoT devices produce and send. This data is collected by backend from Pelion and stored in the database. The Mongo database is structured so, that each user has their own collection, where data about individual devices is stored. There is no upper limit for the collection size, so this system is fine, as long as one device is linked to only one user. The collected data is then accessible even without Pelion connection, but information about the devices

Pelion End-to-End application

themselves can not be retrieved. In figure 2 you can see a JSON representation of a single data-entity in the database.

```
{
  "_id" : ObjectId("5c0d9f76aa8df173bc4c18bf"),
  "payload" : 40.0,
  "ct" : "text/plain",
  "path" : "/3303/0/5700",
  "callback_time" : 1544396662.62475,
  "max-age" : 0,
  "ep" : "016716b754cd00000000000100100039"
}
```

Figure 2. Data entity in MongoDB

The functionality that is provided to the user by the frontend consists of visualization and device manipulation. The user may list all the devices which are attached to the users Pelion account. In case the devices are registered, which means that they are connected to the internet and Pelion, they show as registered in the frontend. Registered devices can be clicked, and their resources will be shown. When the devices resources are listed, the user may view all of them. Resources are split between observables and non-observables, former of which the user can subscribe to. When subscribing to an endpoint or a resource, the backend sends the subscribe-command to Pelion. After subscribing, Pelion starts to send the corresponding data to the backend. This data is then fetched from backend to frontend once per minute, and is then visualized to the user. In figure 3 you may see the flow of usage for a user. Figure 4 shows the device listing. Figure 5 shows the subscribed resources of a device and figure 6 shows the incoming data from a device.

521479S Software project (2018)

Pelion End-to-End application

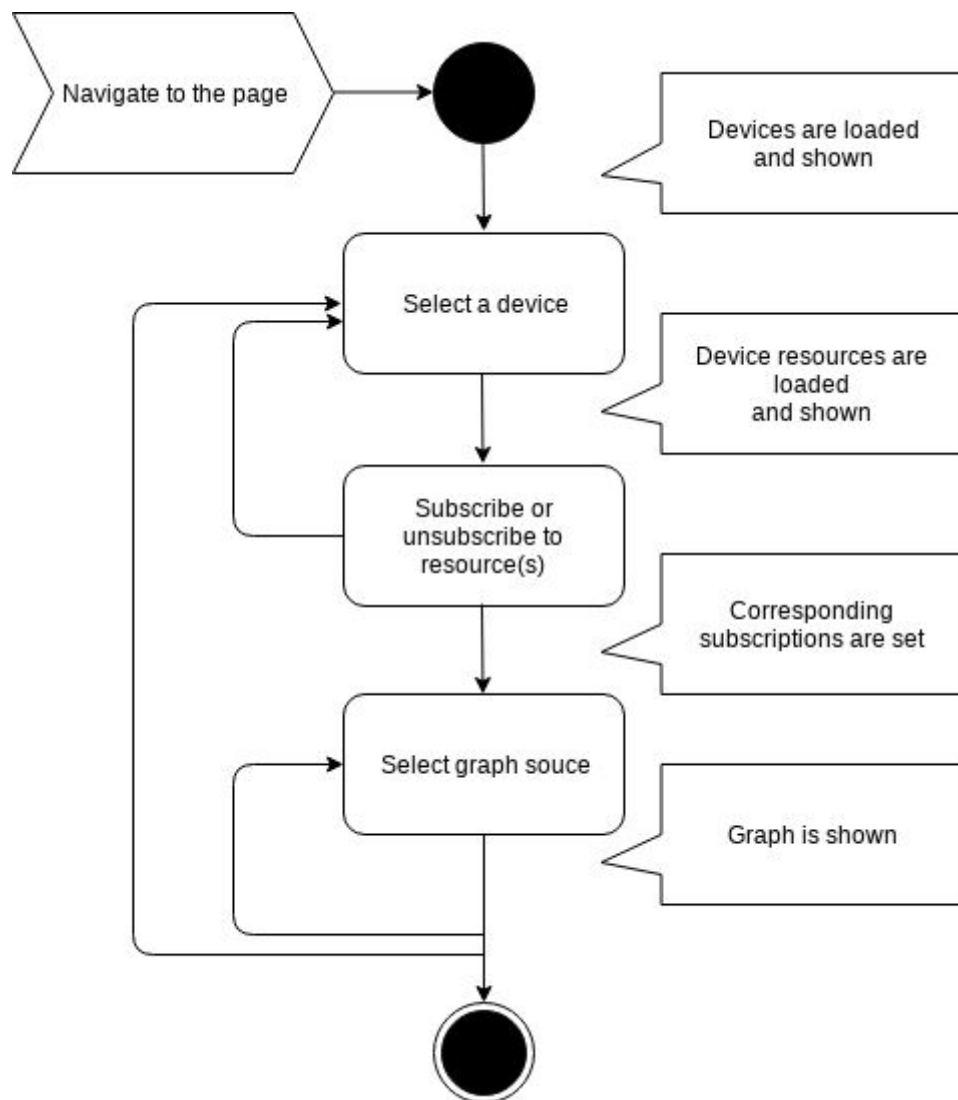


Figure 3. Workflow of the user

Activities **Google Chrome**

Oma Drive - Google... x Yhteistyokumppan... x Google-kalenteri x FINAL_Team-Sig... x SoftwareProject x Final report - Go... x Mbed manageme... x Service API refer... x Pelion-EZE-Proje... x (2) "Jukka kansan... x ... x ... x

Save to RefWork | Other bookmarks

Welcome Mitra Vahida! vahida.mitra@gmail.com
[Devices](#)

<button>Log Devices</button>		<button>Reload Data</button>			
Name	Device type	State	Deployed State	Updated at	Created at
016716b754cd0c00000000000100100039		deregistered	development	2018-12-13T16:12:54.321641Z	2018-11-15T09:32:52.786638Z
016722fd47f0c00000000000100100125		registered	development	2018-12-20T07:04:03.264983Z	2018-11-17T18:44:43.628270Z

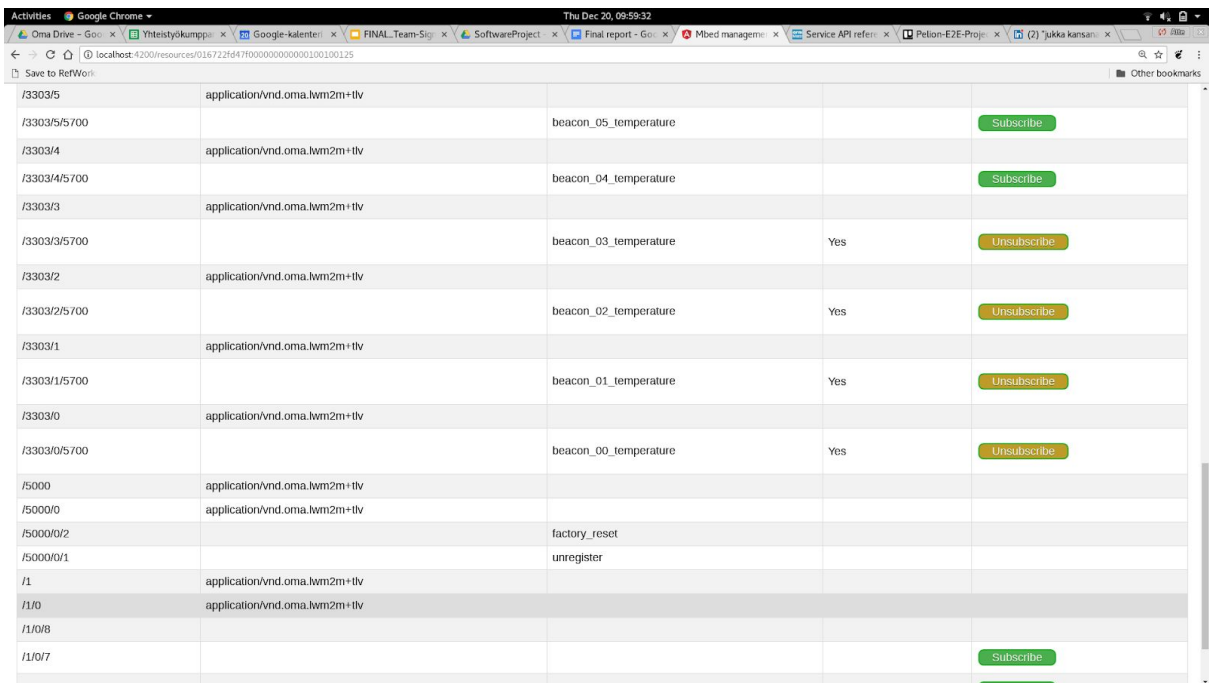
Select the device to draw graphs from:

Refresh

Figure 4. Listed devices

521479S Software project (2018)

Pelion End-to-End application



/3303/5	application/vnd.oma.lwm2m+tlv			
/3303/5/5700		beacon_05_temperature		<button>Subscribe</button>
/3303/4	application/vnd.oma.lwm2m+tlv			
/3303/4/5700		beacon_04_temperature		<button>Subscribe</button>
/3303/3	application/vnd.oma.lwm2m+tlv			
/3303/3/5700		beacon_03_temperature	Yes	<button>Unsubscribe</button>
/3303/2	application/vnd.oma.lwm2m+tlv			
/3303/2/5700		beacon_02_temperature	Yes	<button>Unsubscribe</button>
/3303/1	application/vnd.oma.lwm2m+tlv			
/3303/1/5700		beacon_01_temperature	Yes	<button>Unsubscribe</button>
/3303/0	application/vnd.oma.lwm2m+tlv			
/3303/0/5700		beacon_00_temperature	Yes	<button>Unsubscribe</button>
/5000	application/vnd.oma.lwm2m+tlv			
/5000/0	application/vnd.oma.lwm2m+tlv			
/5000/0/2		factory_reset		
/5000/0/1		unregister		
/1	application/vnd.oma.lwm2m+tlv			
/1/0	application/vnd.oma.lwm2m+tlv			
/1/0/8				
/1/0/7				<button>Subscribe</button>

Figure 5. Listed resources of a single device

521479S Software project (2018)

Pelion End-to-End application

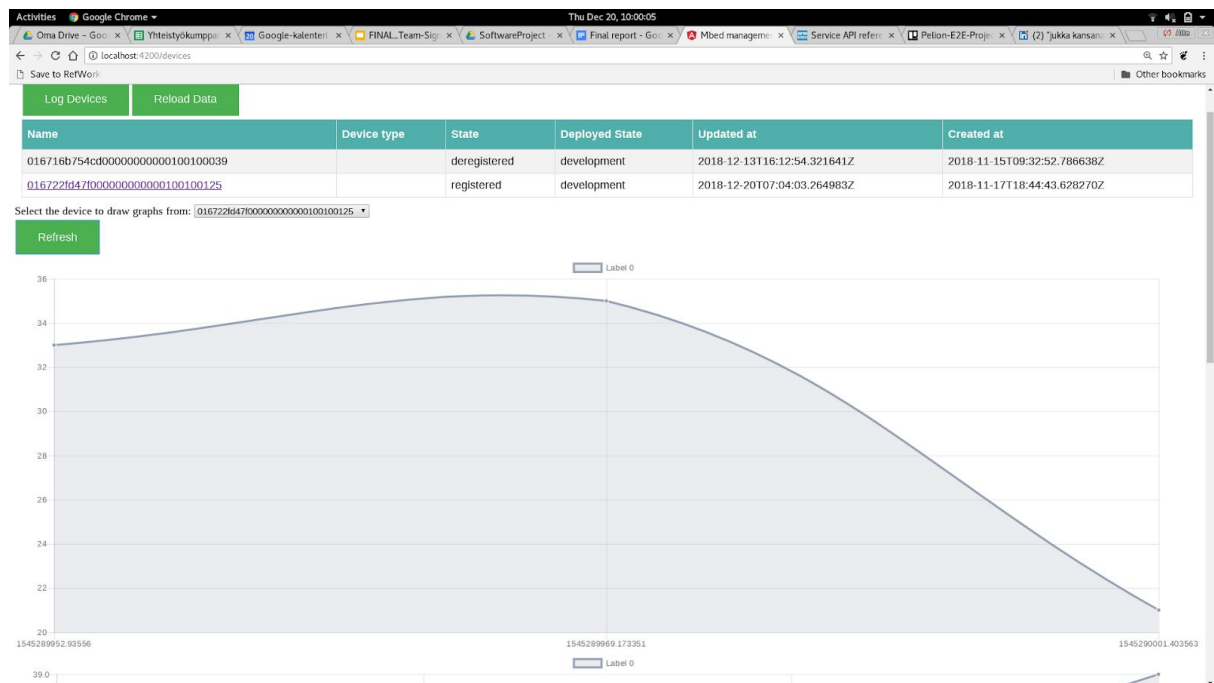


Figure 6. Device data shown in front end

c. Issues

Difficulties were found when trying to expose the service to the web from home. This slowed the development down, but was enabled by installing a modified firmware to my router which enabled port forwarding to a local PC. At that point we were successful in serving the web services through a reverse proxy, thus we were also successful at receiving data from Pelion.

In client side development we experienced some issues with Pelion device management and with the BLE board provided for us by ARM.

There were times when Pelion showed a device as registered when we didn't even have a device powered on. This blocked us from registering a device with the same ID. The problem was solved by deleting the devices in Pelion device management web portal and running a factory reset on the board, which changed the device's ID.

While researching the BLE functionality, we found that a resistor on the BLE board should be moved from one place to another when using it with the K64F-development board, according to [3]. This was however only an inconvenience and the operation was done successfully quite quickly. Also we had to change the pin definition of SPI clock from the BLE driver files and

521479S Software project (2018)

Pelion End-to-End application

update the BLE board firmware according to [4], before the board would work correctly.

With the final client side software, using real BLE connectivity, one issue persists: after around 10 - 20 minutes of runtime, the client hangs or throws a mbed-os specific HardFault exception. Unfortunately we noticed the issue very late in our project and our knowledge of the mbed-os platform is not thorough enough to solve this issue, but our enlightened guess is that the issue should be solvable for someone with more knowledge of the platform.

7. Realised requirements

a. Client

- Getting K64F connected to PC and setting up the development environment
- Internet connection with ethernet and registration to Pelion
- Wi-Fi connectivity
- Simulating BLE beacon devices and data
- BLE connectivity

b. Web app

- Backend and frontend served through a web server
- Backend REST API endpoints established
- Connectivity
- Frontend

c. End-to-end

- Sending and receiving data from the client to the web-app
- Real-time plotting of values from beacon devices in the web-app
⇒ End-to-end connection from BLE beacon device to web-app works

8. Used software development method

We used an agile approach to this project. From the beginning to the end we tried to have a weekly scrum meeting where we talked through the burning issues and how to continue. These sessions were held between the customer and the students. A kanban board in Trello was used to keep track of issues and how they're solved in the project.

9. Testing / acceptance of the software

The MbedOS platform on the client software supports many different testing frameworks. Our choice for this project was to use the MbedOS unittest-framework that is based on

521479S Software project (2018)

Pelion End-to-End application

GoogleTest, which is explained in [5]. MbedOS unit tests are ran on a PC, not the target device. All of our unit test cases passed with the final version of our client software.

Testing of the web application was split into frontend and backend, and were tested with separate frameworks. For frontend, testing was implemented with Pytest, a Python-based testing framework. For backend, tests were implemented with Karma, a JavaScript command line tool. The primary goal of the testing was to confirm that the user can successfully access their Pelion credentials and database using different connection endpoints.

The connectivity of the device was confirmed to function using a simulated web server that loaded the application's source code and applied connectivity tests to it using 10 different user accounts at separate endpoints; every account connected successfully.

Access to the database was confirmed with the user accounts, as was the successful display of the flow of usage, the device's subscribed resources and incoming data flow of usage to the user.

End-to-end testing was done manually with the following steps:

Step 1: Prepare the development board, start the program and wait for the device to register at Pelion.

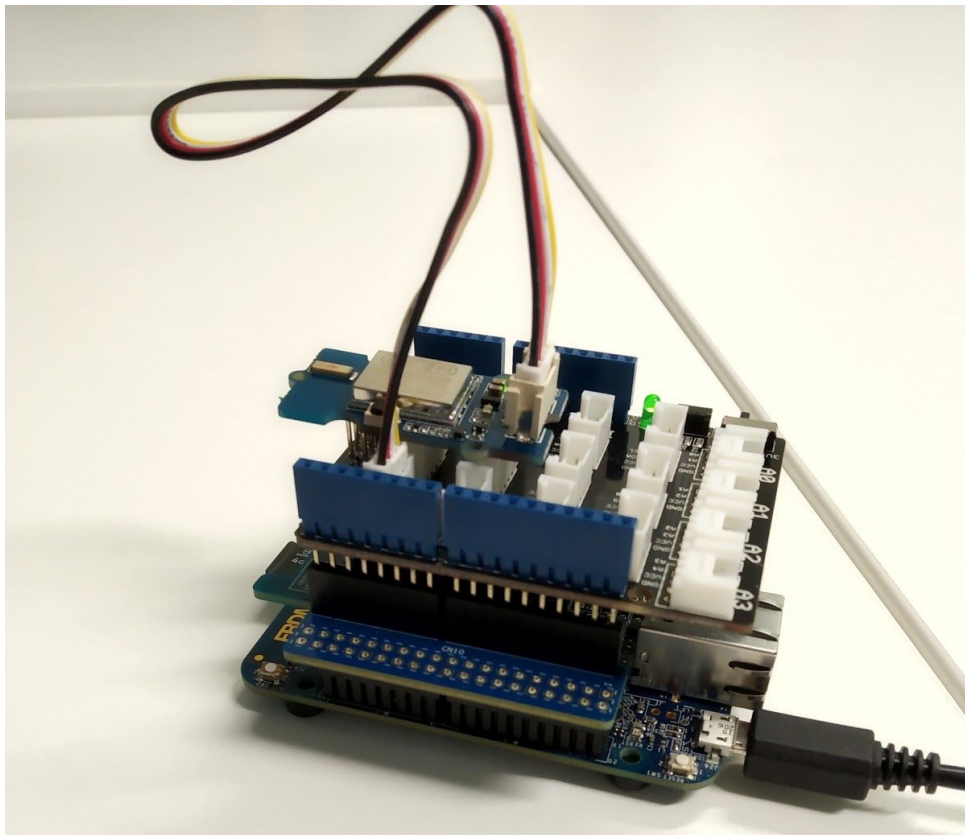


Figure 7. K64F board with Bluetooth-board and WiFi-dongle

521479S Software project (2018)

Pelion End-to-End application

Step 2: Once the device is connected to Pelion, we use a Bluetooth Beacon Simulator application on a mobile phone to send simulated beacon data to the K64F device. The device scans available bluetooth beacons and sends beacon data to Pelion.



Figure 8. Simulated Bluetooth Beacon data

Step 3: After sending data we check the frontend that the device has sent correct information to the backend. There should now be data seen in the frontend as shown in Figure 6.

10. Project timeline

Sep 14th 2018	Course kickoff
Sep 19th 2018	First project meeting
Sep 24th 2018	Web app: Start work
Oct 2nd 2018	Client: K64F up and running and connected to Pelion cloud
Oct 11th 2018	Web app: Properly show user info
Oct 15th 2018	Web app: Show devices from Pelion
Oct 16th 2018	Midterm presentation
Oct 31th 2018	Web app: Show resources for every device
Nov 11th 2018	Web app: Finalize database deployment
Nov 15th 2018	Client: simulated BLE device sensor data and sent to Pelion
Nov 21th 2018	Web app: Set subscription to a resource
Dec 8th 2018	Web app: Draw graphs from device data
Dec 16th 2018	Client: Actual BLE beacon data from Android phone sent to Pelion
Dec 16th 2018	End-to-end: E2E connection verified to be working
Jan 9th 2019	Client: Final UT cases passed

11. Feedback to the course organizers

We think that the course worked as intended since we got to work somewhat closely with people from the industry and contact with the course organizers was (at least in our case) kept to a minimum.