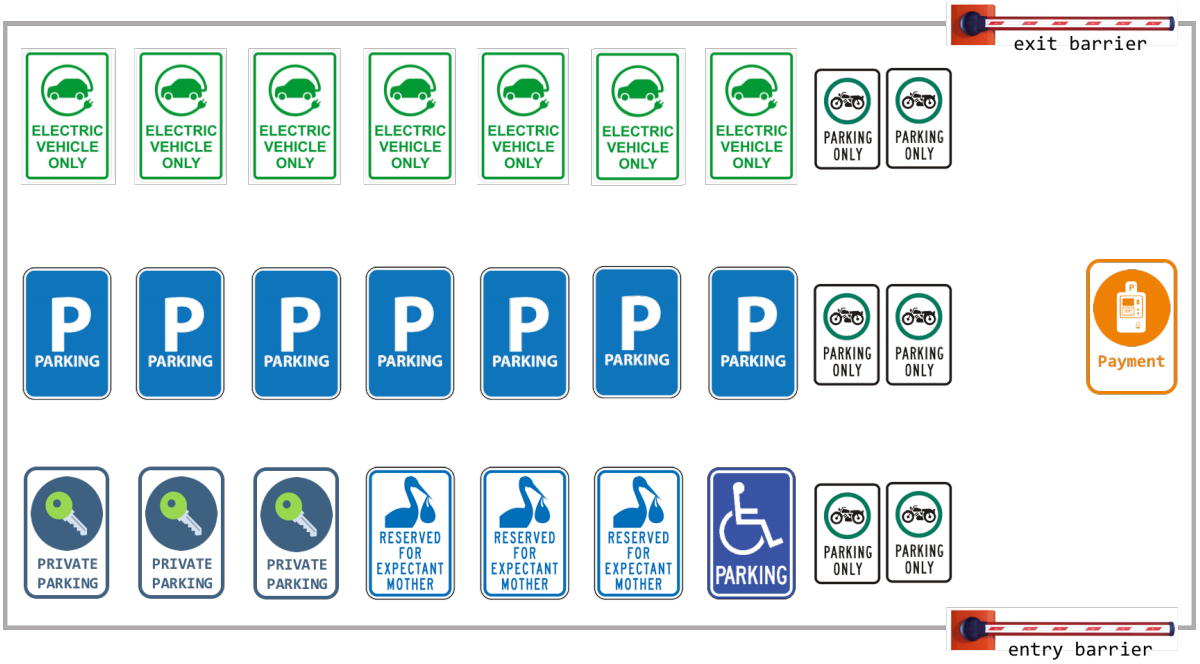
Internet of Things – A.Y. 2016-2017
Final Project

The final project consists of the following steps.

1. Smart Parking (Java language)
After completing the tutorial of CoAP Californium, you should be able to build a CoAP client and a CoAP server, both running on a virtual machine in the PC.

Consider a “Smart Parking” scenario composed by different parking lots, a cash payment box, an entry barrier and an exit barrier, as shown in figure.



Since the considered scenario requires to share information among the different entities, it is suggested to create a Java class in which you can maintain some static objects that are reachable from the different CoAP servers (logically similar to a database), as an example:

```
public class ParkingStorage {
    private static ParkingStorage instance = null;
    public static ParkingStorage instance() {
      if (instance == null) { instance = new ParkingStorage(); }
        return instance;
    }
    public ArrayList<ParkingTicket> tickets = new ArrayList<ParkingTicket>();
}
```

In this way you have the informations that you need everywhere and you can refer to this object with: `ParkingStorage.instance().tickets.<method>()`. In creating the `ParkingTicket` class, feel free to choose if you want to maintain public the internal variables related to the information of the parking, or if

you want to adhere to POJO paradigm, then implementing the getters/setters for each of its internal variables.

1.1. Entry barrier box

The entry barrier is associated with a box that releases you a parking ticket, identified by a random serial number, the parking lot type and date and time of printing. This box is a CoAP server exposing different CoAP resources:

- A CoAP resource, named `emit_ticket`, that need to handle:
  - **POST** requests, in which you pass it the string `<lot_type>` (in which `<lot_type>` can be: `normal`, `private`, `expectant`, `electric`, `hand`, `moto` → e.g. pass `moto` in the body of the **POST** request) and it has to return a string containing the informations related to the emitted ticket. Further, you need to insert a new `ParkingTicket` into the object containing the information related to the tickets list.
  - **GET** requests, returning the total number of emitted tickets.
- A CoAP resource for each lot category, named `tickets_<lot_type>` (e.g., `tickets_moto`), that need to handle:
  - **GET** requests, returning the total number of released tickets for the specific lot type.

Variant 1: adopt JSON format to represent the response of the different CoAP resources maintained by the entry barrier box CoAP.

1.2. Cash payment box

The cash payment box is associated with a CoAP server maintaining an `HashMap` in which is detailed the park cost (€/min) for each type of parking lot (except for `private` type, since the parking lot is private and you don't need to pay anything, and for `hand` type, since it is free) and that need to expose different CoAP resources:

- A CoAP resource, named `parking_payment`, that need to handle:
  - **POST** requests, in which you have to pass it the serial number of your ticket. If the ticket exists, you need to: (i) calculate the dwell time in the parking lot and the total cost of the park, (ii) reply to the client with the details of the ticket, enriched with the dwell time, the total cost of the park and the date and time of payment, and (iii) update, into the tickets list, the payment information of the specific ticket. Conversely, if the ticket does not exist or if it is already paid, you need to inform the client of the occurred situation.
- A CoAP resource, named `paid_lots`, that need to handle:
  - **GET** requests, returning the list of the already paid tickets.
- A CoAP resource, named `unpaid_lots`, that need to handle:
  - **GET** requests, returning the list of the unpaid tickets.

Variant 2: provide the opportunity to pay the ticket with different payment methods (e.g., credit card, debit card, prepaid card). Accordingly to this, handle this new payment method information into **POST** handler and into tickets list object.

1.3. Exit barrier box

The exit barrier is associated with a box that retires the parking tickets. This box is a CoAP server exposing a CoAP resource, named `dismiss_ticket`, that need to handle:

- **POST** requests, in which you have to pass it the serial number of your ticket. If the ticket exists and has been already paid, you need to: (i) reply to the client with a successful message and (ii) delete

the ticket entry from the tickets list. Conversely, if the ticket does not exist or if it is unpaid, you need to inform the client of the occurred issue.
- **GET** requests, returning the total number of dismissed tickets.

1.4. Parking lot

Each parking lot is associated with a CoAP server maintaining some information related to the lot (e.g., name, type, floor level, lot number (printed on the ground) and status (busy/free) of the specific parking lot) and that need to expose a CoAP resource, named `parking_lot`, that need to handle:

- **POST** requests, in which you have to pass it the value "0" if you want to free up the parking lot or "1" if you want to occupy it. If you require to occupy the parking lot, you need to check its status, informing the requester about the correctness of the occupation or the issues related to the attempt to occupy an already busy parking lot.
- **GET** requests, returning the information related to the parking lot.

Variant 3: instead of simply passing the values "0" or "1" to the POST handler, also provide the serial number of your ticket and then save this new information into the CoAP resource. In this case, the **GET** handler need to be enriched (when it responds to CoAP requests) with the information related to the ticket that correspond to the current parking lot.

1.5. Private parking lot

Since you are the owner of a private parking lot, you need to be sure that in this place the only admitted vehicle will be your own one. In this way, you need to implement a CoAP server that works as the one in case **1.4** and that accept an input code, representing your secret parking code, in conjunction with the serial number of the ticket released by the entry barrier box. Further, you need to perform all the checks that you feel necessary to be sure that only your own vehicle is stopping in this private parking lot.

1.6. Electrical parking lot

If you are the owner of an electrical vehicle and you want to recharge it into one of the available electrical parking spots, you need to provide at the entry barrier, in addition to the `<lot_type>`, also the license plate number of your vehicle, and this information needs to be printed on the emitted ticket and stored into the ticket list. You need to provide this additional information also when you place your vehicle inside one of the available electrical parking lots. In addition, since this is a new technology, in order to be secure that these parking events will never be lost, you need to do a **POST** of the received information (from the CoAP server of the electrical parking lot) to a remote CoAP server that maintain a list of parking events related to these electrical parking lots (a unique remote CoAP server to which all the electrical CoAP servers send **POST**s).

1.7. CoAP clients

Implement a single CoAP client for each of the previously exposed parking modules (entry barrier box, exit barrier box, payment box, some parking lots), considering whether it is necessary/useful to mutate from non-observable to observable some of the previously GET-enabled CoAP resources, if you feel the need. Try to do these CoAP clients as configurable as possible, since they need to be runnable also outside your PC (suggestion: provide them the possibility to receive parameters).

2. Final report

Describe what you have done (architecture design and software scripts) that you have developed in the previous steps. Whenever possible (e.g., step 1), insert code snapshots and/or results related to the measurements of total free/busy time of parking lots among the different days.

3. Final delivery of the developed project

You are requested to deliver the final report, in PDF format, and the complete developed code, in a ZIP archive, named `surname_name_studentid.zip`.
The archive should contain a folder named `surname_name_studentid`, containing in turn these files.
The ZIP archive should be sent through a unique e-mail to Dr. Luca Davoli (**luca.davoli@unipr.it**) and in CC to Prof. Gianluigi Ferrari (**gianluigi.ferrari@unipr.it**).

Be sure to send the ZIP archive to both addresses together