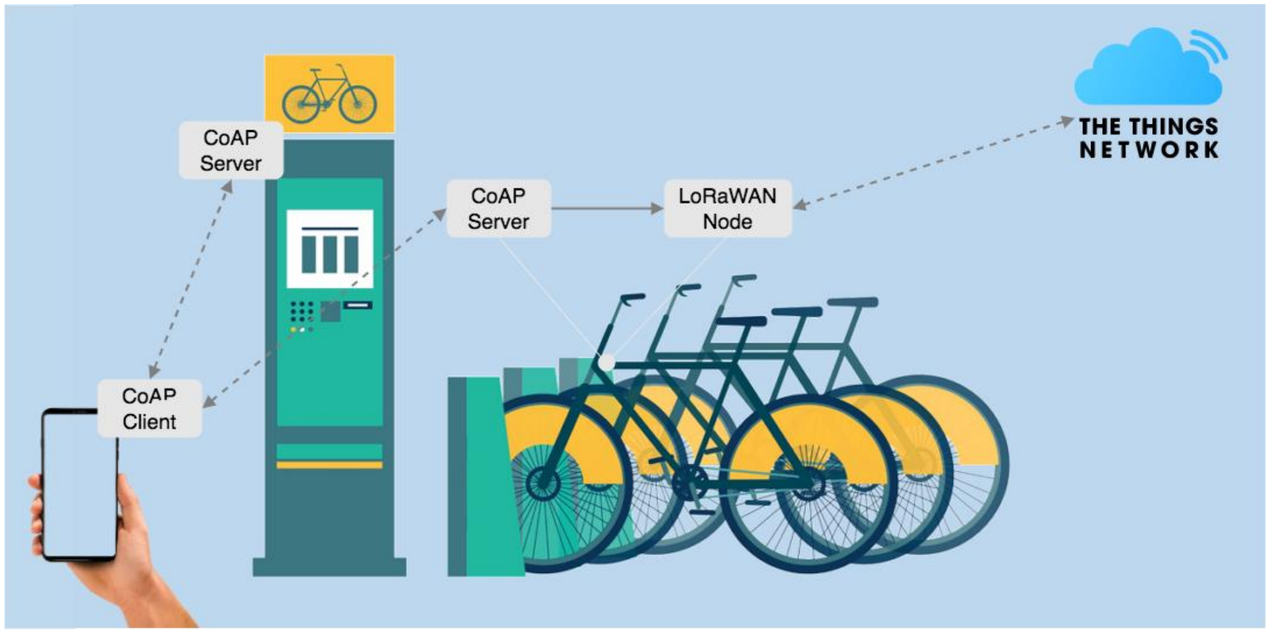
A Bike Sharing Service

Internet of Things Project

Giulia Oddi

Outline

1. Goals
2. Bike Entity
3. Exchange Schema
4. Architecture
   - Cyclist Client
   - Manager Client
   - Docking Station Server
   - Bike CPU Server
5. Conclusions



Goals

- Realization of a **bike sharing service**.
- Modelling part of the system in a **simulated environment.**
- Using Java:
  - **CoAP**
  - **Californium** framework.
- Cyclists shall be able to **rent** a bike from a single Docking Station.
- **Smart bikes** controlled by an on-board embedded CPU.

Bike Entity

- In order to define and use a **bike object.**
- Following the Plain Old Java Object (POJO) paradigm.
- Features (private variables):
  - `id` = the 10-characters bike id
  - `fiscalCode` = the fiscal code of the cylist (default null = no rented)
  - `available` = boolean value for the availability of the bike (default true)
  - `latitude` = double value for the latitude of the bike (default 44.7653076)
  - `longitude` = double value for the longitude of the bike (default 10.3067761)
  - `frontBrakePress` = boolean value that indicates if the front brake is pressed (default false)
  - `rearBrakePress` = boolean value that indicates if the rear brake is pressed (default false)
  - `gearbox` = integer value that represents the 8-gearbox (default 4)

Exchange Shema

- The exchange schema adopted for representing the information to be sent through CoAP POST requests is the **JSON format**.

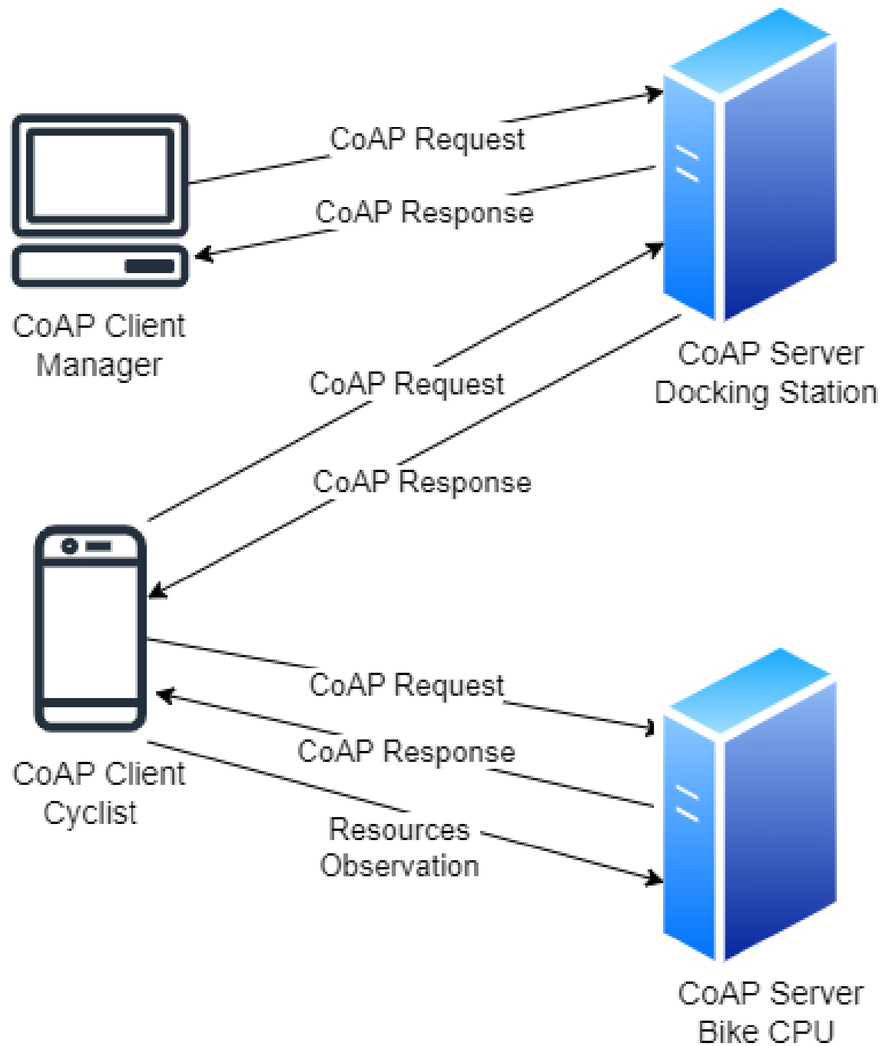
```
{
   "id":"B0000-5665",
   "fiscalCode": null,
   "available": true,
   "latitude": 44.7653076,
   "longitude": 10.3067761,
   "frontBrakePress": false,
   "rearBrakePress": false,
   "gearbox": 4
}
```

Example: JSON format for a Bike entity

- Using the `gson`*:* a Java serialization/deserialization library by Google:
  - Added the `.jar` file in `/libs` directory.
  - `import com.google.gson.Gson;`

Architecture (1/5)

- The system is composed by 2 CoAP servers and 2 clients



Architecture (2/5)

Cyclist Client

- A cyclist is identified by the 16-character **fiscal code**, that he must give to the application at the begin (`Scanner class`)**.**
- Then, a cyclist can:

    A) Ask for the list of available bikes

    B) Ask to rent an available bike

    - Change the gear (from 1 to 8)
    - Press or release the brakes
    - Check all bike's information
    - End the trip

    E) Exit

- In order to manage all the possibilities, the cyclist has 8 **CoAP Clients**.

Architecture (3/5)

Manager Client

- In order to manage the infrastructure, there is also a **manager.**
- The manager can:

  A) Ask for the list of available bikes

  B) Ask for the list of rented bikes

  C) Ask for the identity of the owner of a certain bike

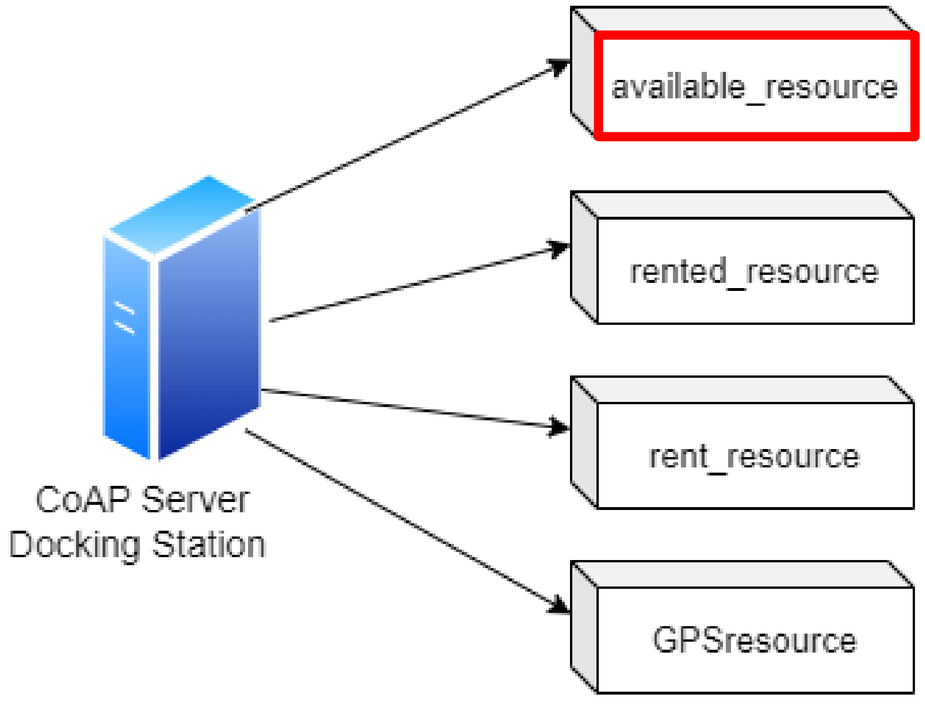
  E) Exit

- To manage all the possibilities, the manager has 3 **CoAP Clients**.

Architecture (4/5)

Docking Station Server

- The Docking Station as been modelled as **a CoAP server.**
- It manages a set of 10 bikes.



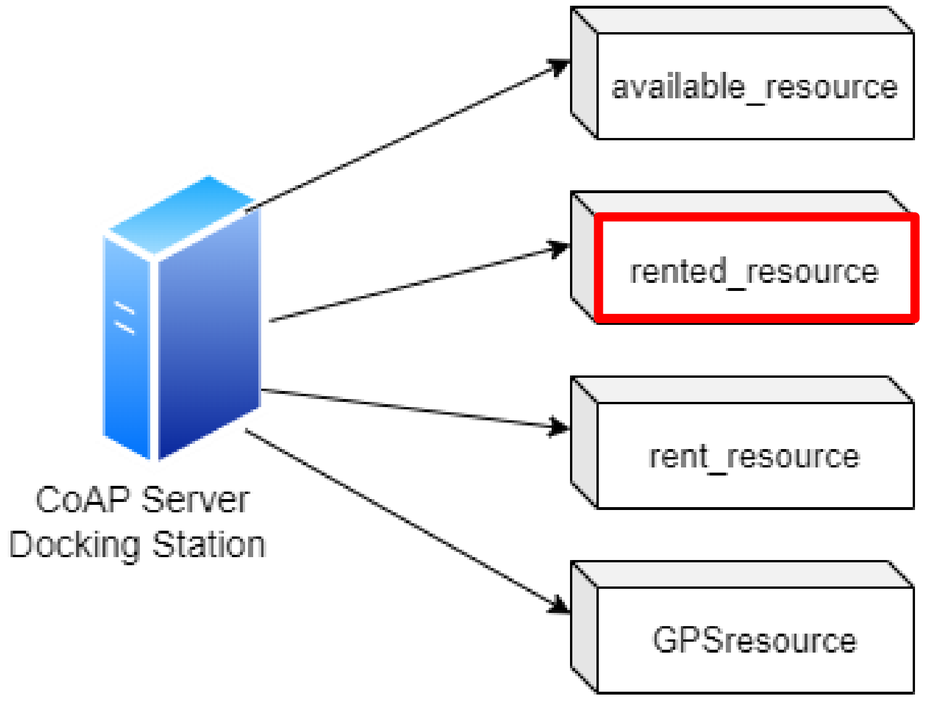
AvailableBikeResource

- It returns the list of the **available** bikes, handling a GET request.
- If no bike available, it returns a `ResponseCode.NOT_FOUND` response.

Architecture (4/5)

Docking Station Server

- The Docking Station as been modelled as **a CoAP server.**
- It manages a set of 10 bikes.



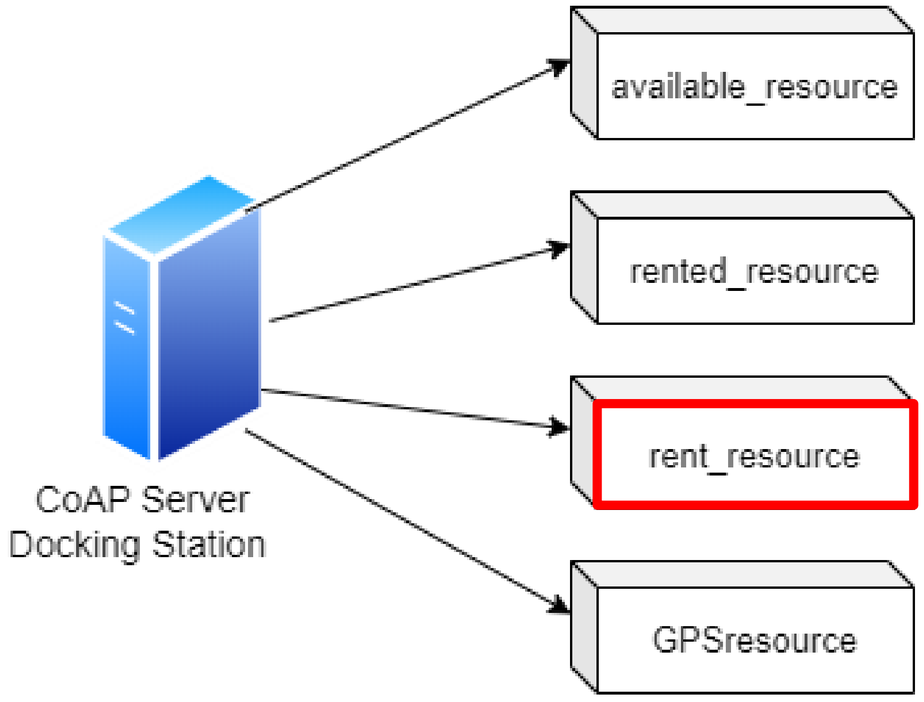
RentedBikeResource

- It returns the list of **rented** bikes, handling a GET request.
- If no bike rented, it returns a `ResponseCode.NOT_FOUND` response.

Architecture (4/5)

Docking Station Server

- The Docking Station as been modelled as **a CoAP server.**
- It manages a set of 10 bikes.



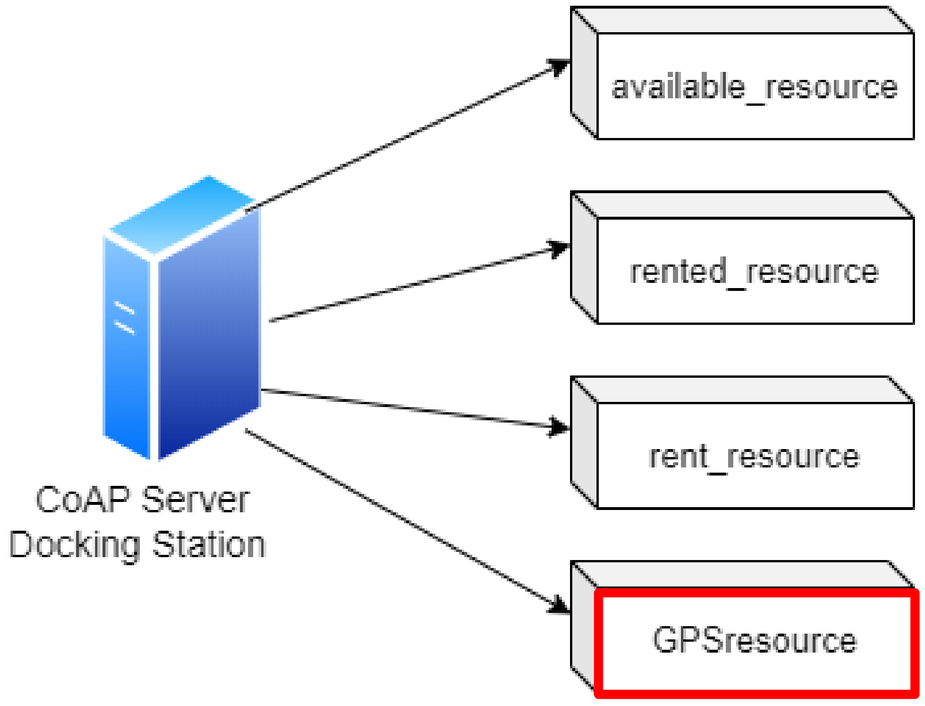
RentResource

- It allows a cyclist to **rent** a bike, handling a POST request.
- It allows the cyclist to end the trip and release the bike.
- If the bike isn't available, it returns a `ResponseCode.BAD_REQUEST` response.

Architecture (4/5)

Docking Station Server

- The docking station as been modelled as **a CoAP server.**
- It manages a set of 10 bikes.



GPSCoordinatesResource

- It allows a bike to save its **GPS coordinates**, handling a POST request.
- If the bikeID isn't correct, it returns a `ResponseCode.NOT_FOUND` response.

Architecture (5/5)

Bike CPU Server

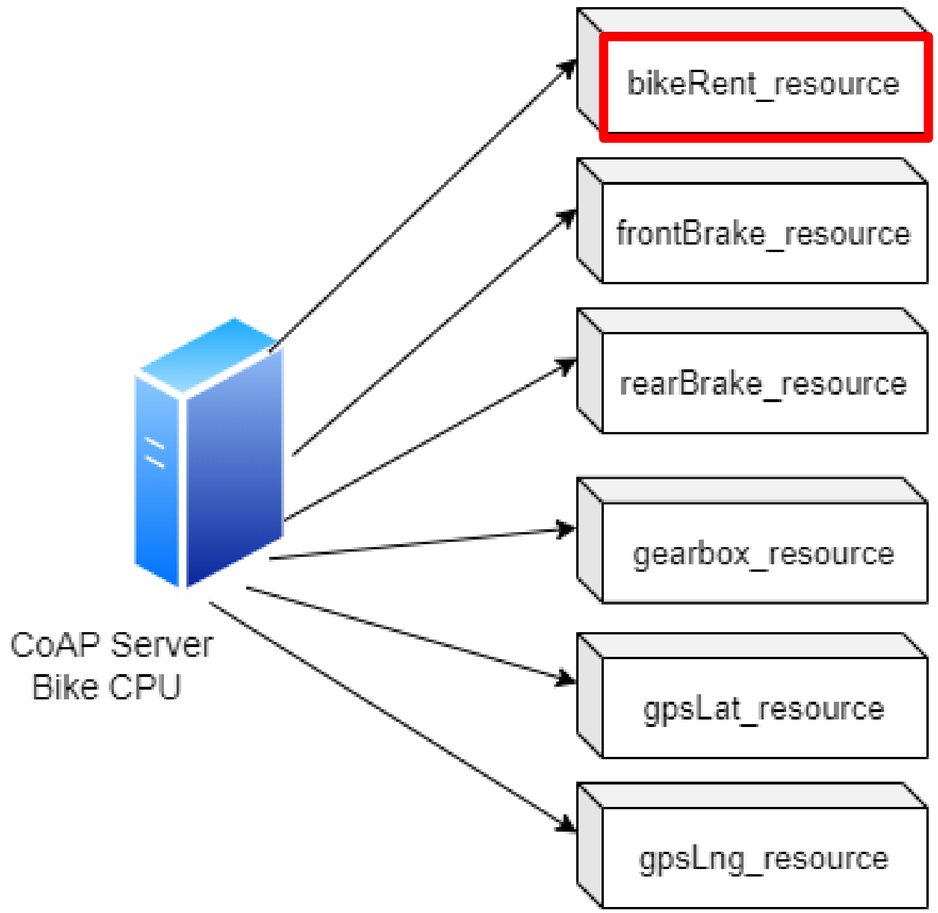
- Each bike's embedded CPU can be modelled as **CoAP server** in charge of managing all relevant bike's information.
- Each bike is characterized by a unique 10-characters ID, that has a standard **format**:

```
String id = String.format("B000%s-%s",i,port);
```

so, the **port** used to connect clients to the correct bike CPU server is given by the last 4 number of the bike ID.

- Each bike's CPU server has a **CoAP client** that sends POST requests to the Docking Station server in order to update GPS coordinates, every 20 seconds and with a variation of ±0.001.

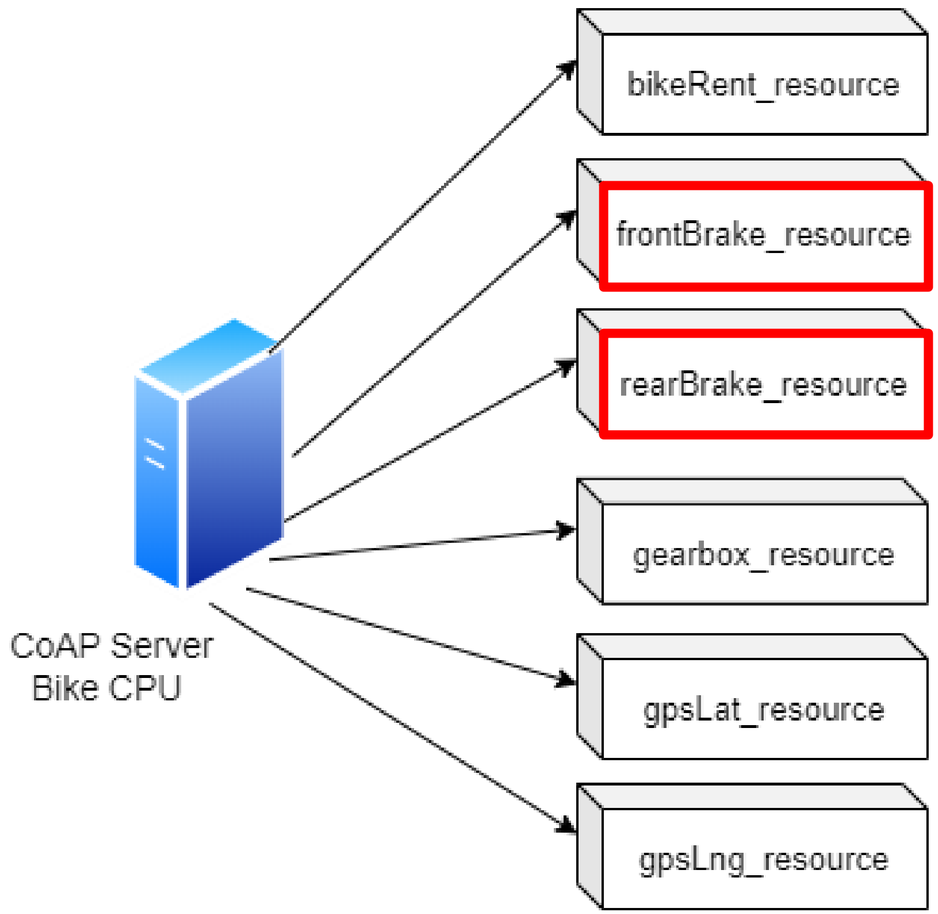
Architecture (5/5)
Bike CPU Server



BikeRentResource

- It allows a cyclist to **rent** the bike, handling a POST request.
- If the bike is already rented, it returns a `ResponseCode.BAD_REQUEST` response.
- It allows also the cyclist to end the trip and release the bike.
- It returns to the manager the **identity** of the cyclist, handling a GET request.
- If the bike isn't rented, it returns a `ResponseCode.BAD_OPTION` response.

Architecture (5/5)
Bike CPU Server



FrontBrakeResource

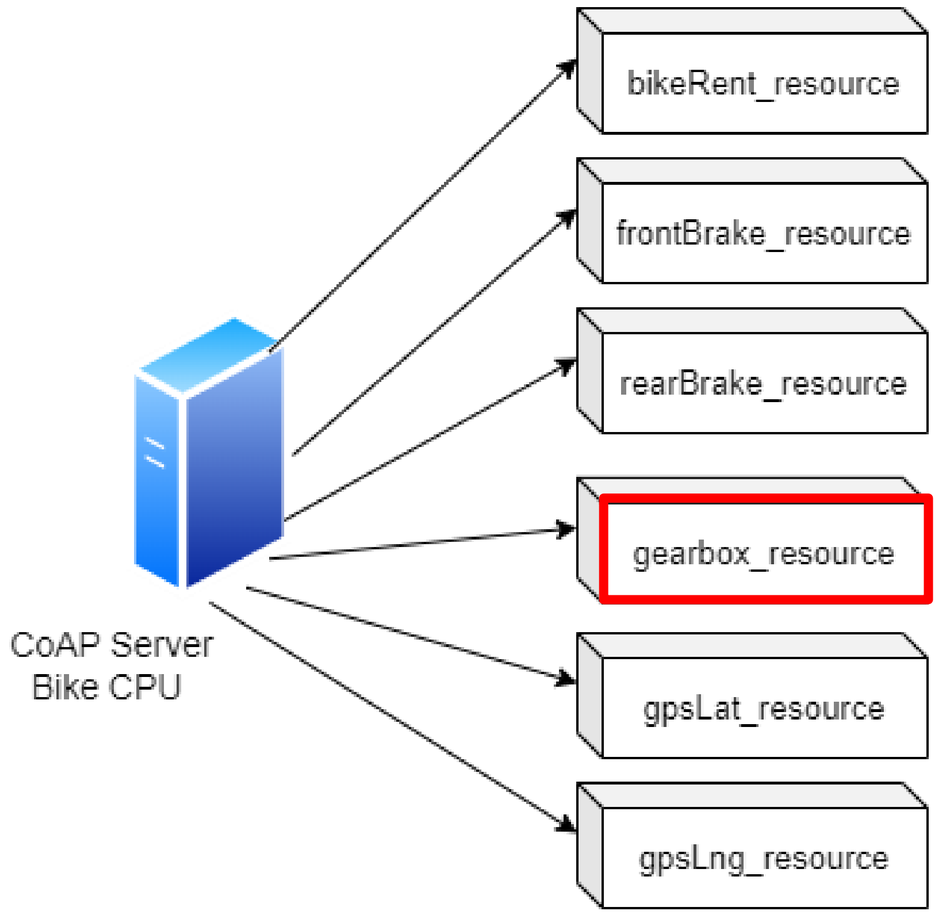
- It allows the cyclist to **press** or **release** the front brake lever, handling a POST request.
- **Observable** resource.
- It returns to the cyclist the current status of the front brake, handling a GET request.

RearBrakeResource

- It allows the cyclist to **press** or **release** the rear brake lever, handling a POST request.
- **Observable** resource.
- It returns to the cyclist the current status of the rear brake, handling a GET request.

Architecture (5/5)

Bike CPU Server

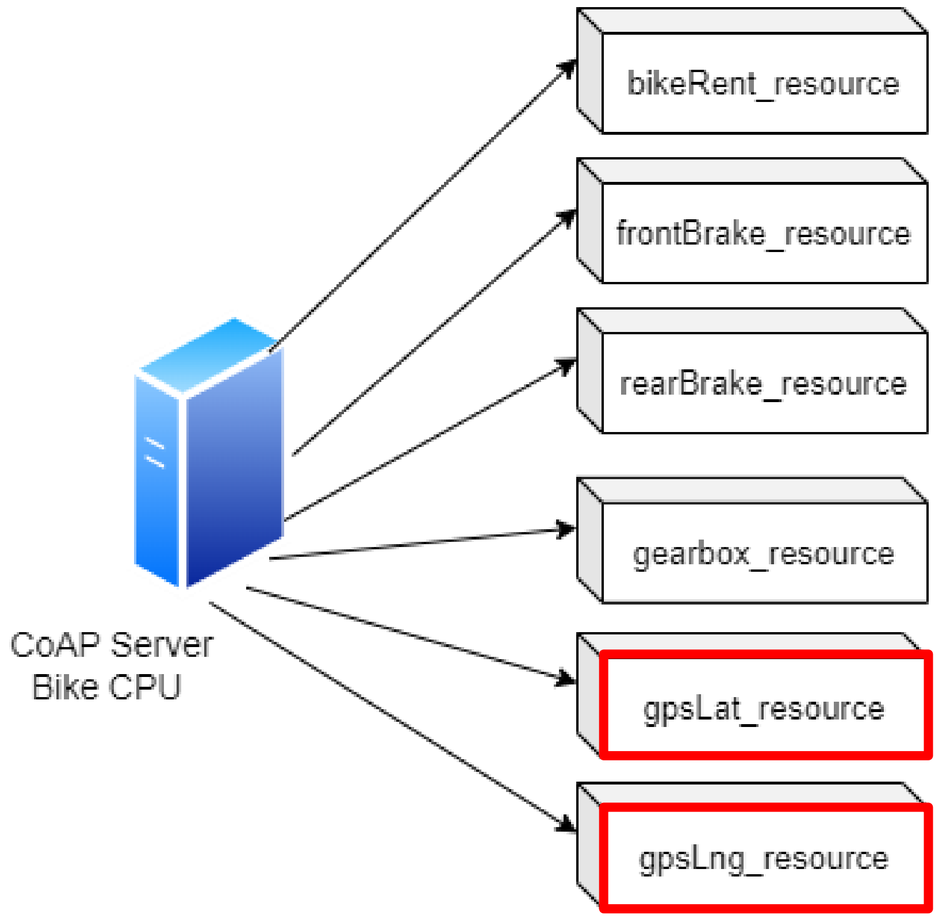


GearboxResource

- It allows the cyclist to change the **gear** from the handlebar, handling a POST request.
- If the cyclist tries to decrease the 1st gear or increase the 8th gear, it returns a `ResponseCode.FORBIDDEN` response.
- **Observable** resource.
- It returns to the cyclist the current status of the gear, handling a GET request.

Architecture (5/5)

Bike CPU Server



GPSlatResource

- It returns to the cyclist the current GPS latitude, handling a GET request.
- **Observable** resource.

GPSlngResource

- It returns to the cyclist the current GPS longitude, handling a GET request.
- **Observable** resource.

Conclusions

- In **conclusion**, the project simulates an infrastructure for bike sharing located in the Campus (Università di Parma).
- There is a **Docking Station** that administrates the bikes.
- Each **bike** is controlled by an embedded CPU, that manages all the relevant bike's information.
- The **cyclist**, after entering his fiscal code, can rent a bike and control it.
- The **manager** can control the status of the bikes.

Thank You For Your Attention!