

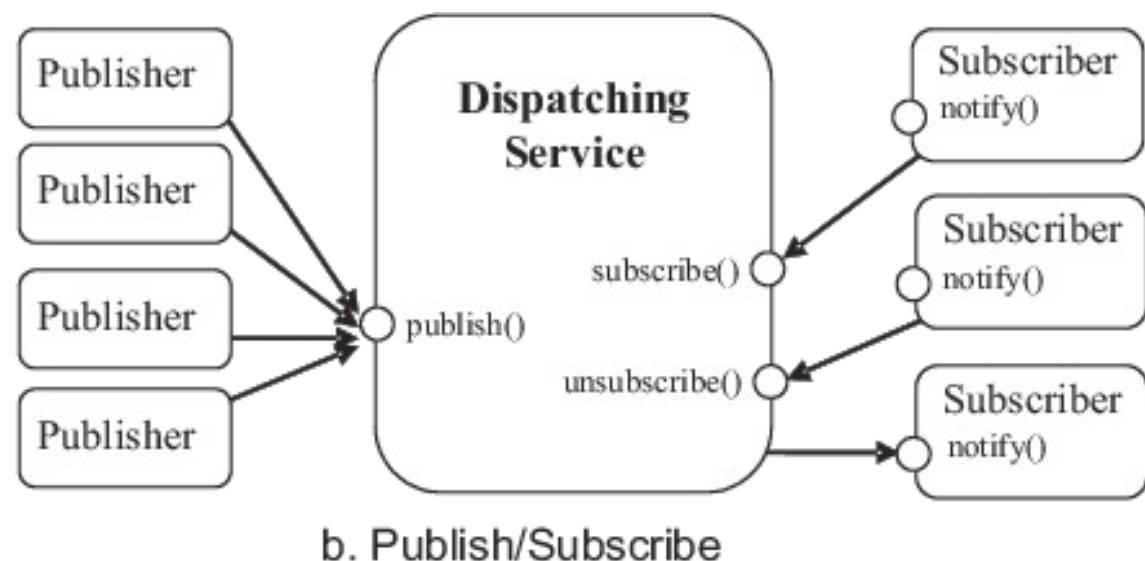
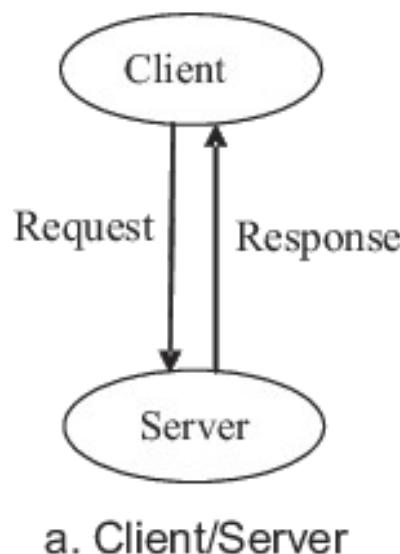


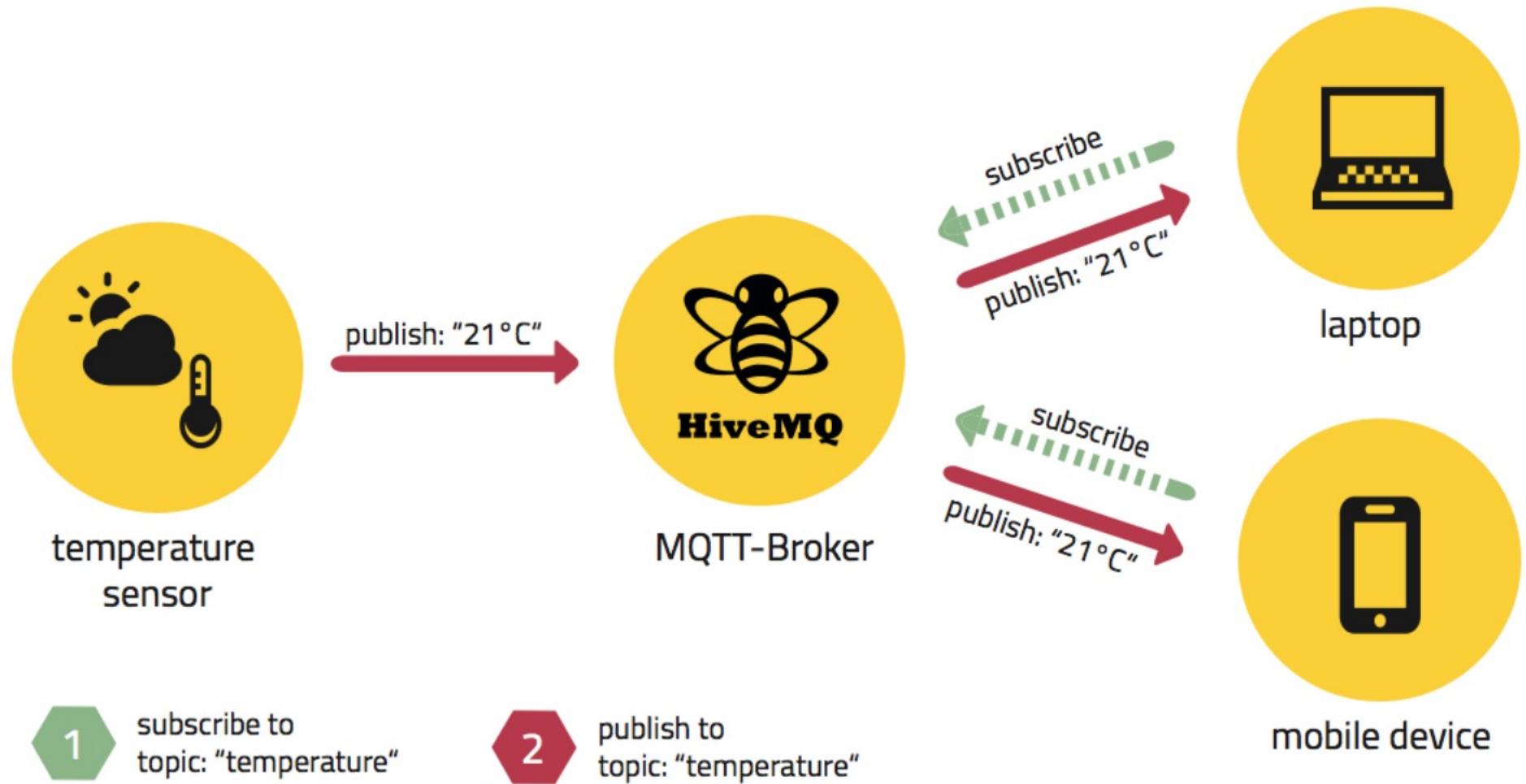
Message Queuing Telemetry Transport

**Introduction to MQTT and its properties.
Example in Java with the Eclipse Paho library
and the Mosquitto broker.**

- "A machine-to-machine (M2M) connectivity protocol, extremely simple and lightweight **publish/subscribe** messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks."
 - well suitable for the IoT
- Message Queuing Telemetry Transport
 - <http://mqtt.org>
- Invented in 1999 by Andy Stanford-Clark (IBM) and Arlen Nipper (now Cirrus Link)
- Version 5 is an OASIS standard

- Request/response model
 - es. HTTP
- Publish/subscribe model
 - es. MQTT





- It implements a brokered publisher/subscriber pattern by relying on TCP
- Broker
 - the central communication point
 - it is in charge of dispatching all messages between the senders (publishers) and the rightful receivers (subscribers)
- Publisher
 - the sender of one or more messages
 - each message has a *topic*
- Subscriber
 - the receiver of one or more messages
 - each subscriber listens to one or more *topics*

- The publisher and subscribers don't know about the existence of one another
- The clients don't know each other, but only the message broker, which filters all incoming messages and distributes them to the correct subscribers
- Therefore, we have:
 - space decoupling, publisher and subscriber do not need to know each other (e.g., by IP address and port)
 - time decoupling, publisher and subscriber do not need to be connected at the same time
 - synchronization decoupling, operations on both components are not halted during publishing or receiving messages

- The routing information for the broker
- A simple UTF-8 string that can have more hierarchy levels – separated by a slash
- E.g., a sample topic for sending temperature data of a living room could be **home/living-room/temperature**
- The subscriber can subscribe to the exact topic or use a wild card

- +
 - A single level wild card that only allows arbitrary values for one hierarchy
- #
 - a multilevel wild card that allows to subscribe to all the underlying hierarchy levels
- Examples:
 - home/+/temperature
 - home/#

- Each MQTT message is published with one of three Quality of Service (QoS) level
- Each QoS level is associated with different guarantees with regards to the reliability of the message delivery
- Both client and broker may provide additional persistence and redelivery mechanisms to increase reliability in case of network failures, restarts of the application, and other unforeseen circumstances

QoS Level	Description
0	At most once delivery. The sender tries with best effort to send the message and relies on the reliability of TCP. No retransmission takes place.
1	At least once delivery. The receiver will get the message at least once. If the receiver does not acknowledge the message or the acknowledgement gets lost on the way, it will be resent until the sender gets an acknowledgement. Duplicate messages can occur.
2	Exactly once delivery. The protocol makes sure that the message will arrive exactly once at the receiver by using a four step handshake. This increases communication overhead but is the best option when neither loss nor duplication of messages are acceptable.

- Last Will and Testament (LWT)
 - A LWT message can be specified by a publisher when connecting to a MQTT broker.
 - If that client doesn't disconnect gracefully, the broker sends out the LWT message on behalf of the client when connection loss is detected.
- Retained Messages
 - Each message may be sent as a retained message, i.e., its last known good value. It persists at the MQTT broker for the specified topic.
 - Every time a new client subscribes to that specific topic, it will instantly receive the last retained message on that topic.

- Clean/Durable Session
 - On connection, a client may set the "clean session" flag
 - If clean session is true, then all subscriptions will be removed for the client when it disconnects
 - If clean session is set to false, then the connection is treated as durable: when the client disconnects, any subscriptions it has will remain and any subsequent QoS 1 or 2 messages will be stored until it connects again in the future (with the same client id)

Broker	Description
Eclipse mosquitto	An open source MQTT broker written in C. It fully supports MQTT 3.1 and MQTT 3.1.1 and is very lightweight. Due to its small size, this broker can be used on constrained devices.
HiveMQ	A scalable, high-performance MQTT broker suitable for mission critical deployments. It fully supports MQTT 3.1 and MQTT 3.1.1 and has features like websockets, clustering, and an open-source plugin system for Java developers.
IBM WebSphereMQ	A commercial message-oriented middleware by IBM that fully supports MQTT.

- Binary available for multiple OS, often directly in the system packages manager
 - <https://mosquitto.org/download/>
- Test brokers available at
 - <http://test.mosquitto.org>
- UniUPO broker
 - smartcity-challenge.edu-al.unipmn.it (IP: 193.206.52.98) porte standard di mqtt
 - user pissir, passwd pissir2020



Library	Description
Eclipse Paho	Paho clients are among the most popular client library implementations, available for several programming languages (Java and C# included).
M2MQTT	M2MQTT is an MQTT client library for .NET and WinRT (C# only).
Fusesource MQTT Client	The Fusesource MQTT client is a Java MQTT client with 3 different API styles: Blocking, Future-based, and Callback-based.

- Open source implementation of the MQTT messaging protocol
- <https://www.eclipse.org/paho/>
- Available for several programming languages
- Latest version of the Java library:
 - 1.2.1

- Using gradle

```
compile  
"org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.2.5"
```

- or Maven

```
<dependency>  
  <groupId>org.eclipse.paho</groupId>  
  <artifactId>org.eclipse.paho.client.mqttv3</artifactId>  
  <version>1.2.5</version>  
</dependency>
```

Tool	Description
MQTT.fx	A JavaFX application with a clean interface: http://mqttfx.jensd.de . For Windows, macOS, and Linux.
mqtt-spy	An open source desktop and command line utility intended to help you with monitoring activity on MQTT topics: https://github.com/eclipse/paho.mqtt-spy . For Windows, macOS, and Linux.
MQTTLens	A Google Chrome extension, which connects to a MQTT broker and is able to subscribe and publish to MQTT topics: https://chrome.google.com/webstore/detail/mqttlens/hemojaaeigabkbcookmlgmdigohjobjm?hl=en

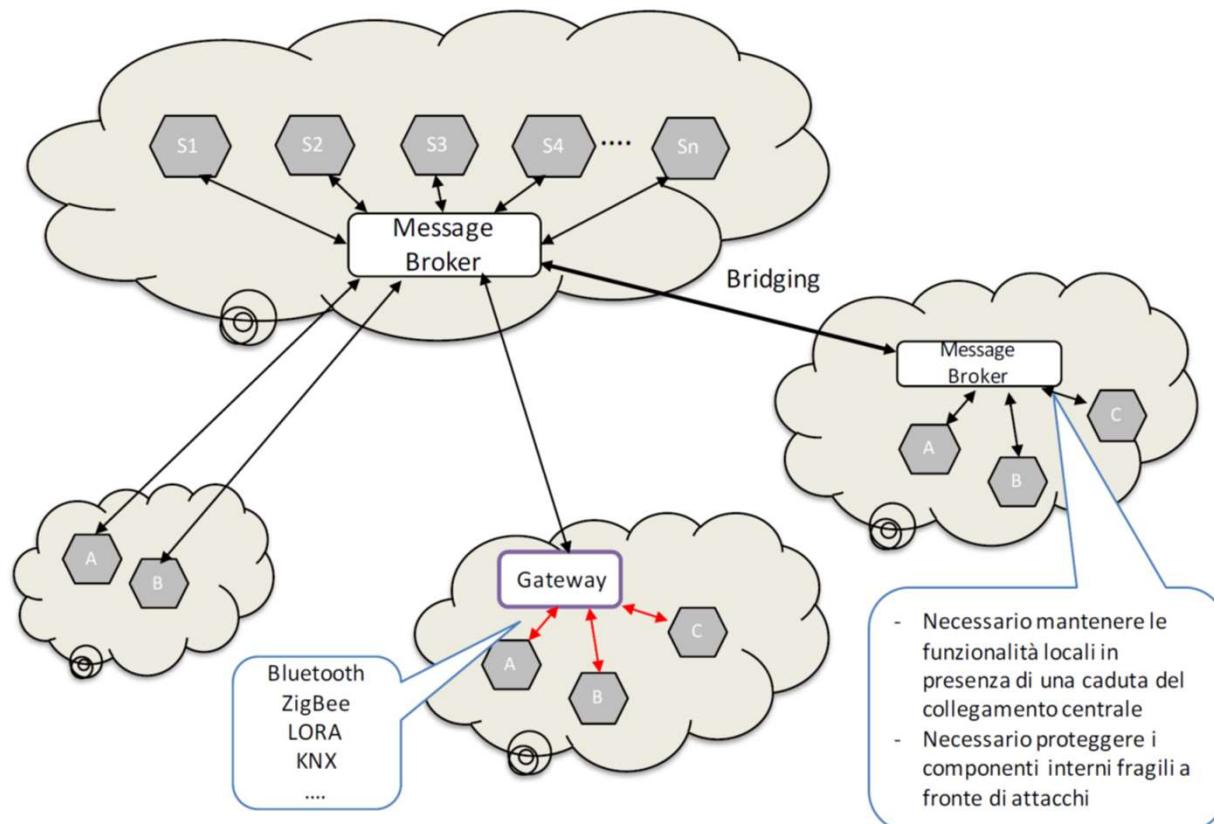
- MQTT
 - <http://mqtt.org>
- MQTT wiki
 - <https://github.com/mqtt/mqtt.github.io/wiki>
- Mosquitto man page
 - <https://mosquitto.org/man/mqtt-7.html>
- MQTT 101
 - <https://www.hivemq.com/blog/how-to-get-started-with-mqtt>
- MQTT Essentials
 - <https://www.hivemq.com/mqtt-essentials/>

- This work is licensed under the Creative Commons “Attribution-NonCommercial-ShareAlike Unported(CC BY-NC-SA 4.0)” License.
- You are free:
 - to Share-to copy, distribute and transmit the work
 - to Remix-to adapt the work
- Under the following conditions:
 - Attribution-You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
 - Noncommercial-You may not use this work for commercial purposes.
 - Share Alike-If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.
- To view a copy of this license, visit
 - <https://creativecommons.org/licenses/by-nc-sa/4.0/>
- This work is a derivative of MQ Telemetry Transport by Luigi De Russis



Esercitazione MQTT

Tipica Architettura IoT



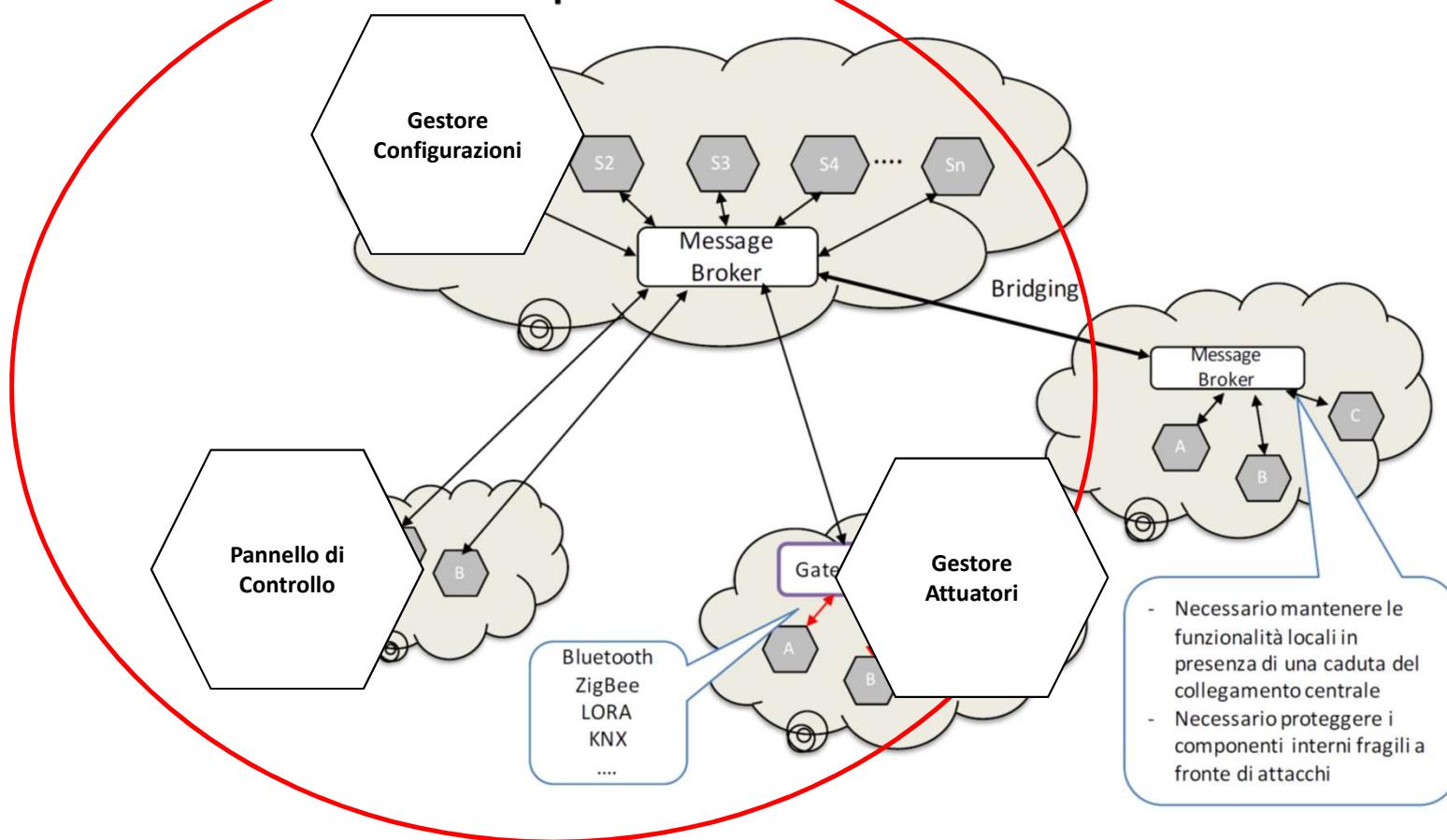
Prototipo di sistema di controllo di attuatori

- Un pannello di comandi che può essere utilizzato dall’utente per azionare degli attuatori e verificarne lo stato agendo su pulsanti senza conoscere il dettaglio di funzionamento degli attuatori
- Un sistema di interfacciamento verso gli attuatori che può inviare comandi a questi ultimi e richiedere agli stessi lo stato
- Un sistema che gestisce la configurazione: tiene traccia della associazione tra ciascun «pulsante» (virtuale) del pannello di comandi con un determinato attuatore fisico.

Visibilità dei componenti

- I tre componenti possono raggiungere internet ma sono eseguiti su host che non hanno un IP pubblico
- L'uso di un broker con IP pubblico risolve il problema di collegare i tre componenti (noi inizialmente eseguiremo i tre componenti e il broker su uno stesso host, ma basterà cambiare alcuni parametri di configurazione per poterli sperimentare in uno scenario distribuito)

Tipica Architettura IoT



Strutturazione dei topic e protocollo di interazione

- Componente 1: pannello di controllo
- Componente 2: gestore attuatori
- Componente 3: gestore delle configurazioni

Tabella:
Switch1 – Attuatore3
Switch2 – Attuatore1
Switch3 – Attuatore 5

\ Topic Comp.	switch	Stato_switch	attuatori	Stato_attuatore
Comp. 1	Pub	Sub		
Comp. 2			Sub	Pub
Comp. 3	Sub	Pub	Pub	Sub

Funzionamento del sistema

Premessa: il configuratore conosce l'elenco dei pulsanti presenti sul pannello di controllo, l'elenco degli attuatori e contiene la tabella delle associazioni pulsante-attuatore (un topic switch/{id_pulsante} per ogni pulsante e attuatori/{id_att} per ogni attuatore, oppure un unico topic switch e un topic attuatori e poi id del pulsante/attuatore nel payload)

Il configuratore si mette in ascolto su switch (o su switch/#) e su questo topic può ricevere dei comandi ACCENDI / SPEGNI (più id_pulsante se non è già nel topic)

Quando riceve un messaggio di questo tipo, sulla base della tabella di associazioni switch-attuatori, invia sul topic attuatori (o su attuatori/{id_att}) un comando GO / STOP

Periodicamente il sistema di interfacciamento verso gli attuatori pubblica sul topic Stato_attuatori/{id_att} lo stato (ON / OFF) degli attuatori di sua competenza.

Tutte le volte che riceve lo stato dagli attuatori il configuratore pubblica sul topic Stato_switch/{id_pulsante} (dove id_pulsante è preso dalla tabella di associazioni switch-attuatori) ACCESO / SPENTO: questo permette al pannello di controllo di avere un riscontro dello stato degli attuatori che può comandare attraverso la visualizzazione dello stato dei pulsanti.

Osservazioni

- Questo sistema permette di disaccoppiare i comandi del pannello di controllo dagli attuatori fisici e permette di
 - avere più pannelli di controllo che gestiscono gli stessi attuatori
 - mantenere un unico punto di configurazione (ed eventualmente replicarlo nel caso in cui il numero di pannelli/attuatori dovesse crescere, per garantire la scalabilità),
 - sostituire un attuatore con un altro (per esempio in caso di guasto) potendo gestire facilmente la riconfigurazione
- Occorre progettare opportunamente la gerarchia dei topic per esempio per raggruppare i pulsanti in base alle stanze: se i pulsanti sono interruttori di luci nella casa potremmo avere casa/cucina/{id_int_cucina} casa/salotto/{id_int_salotto} oppure si possono separare gli attuatori in base alla posizione e/o in base al tipo o al protocollo di interfacciamento.

<https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/>

Demo – Pannello di controllo

- Per una dimostrazione pratica dell'esercizio realizzato si può pensare ad una semplice applicazione con interfaccia testuale a menu che mostra lo stato corrente e mette a disposizione diverse azioni:

STATO: 1 acceso, 2 spento

0- Aggiorna visualizzazione stato

1- Accendi 1

2 - Spegni 1

3 - Accendi 2

4 - Spegni 2

Demo - Attuatori

- Il gestore degli attuatori può semplicemente visualizzare lo stato di tutti gli attuatori di sua competenza stampando sulla console, ogni volta che c'è un cambiamento di stato, il nuovo stato di tutti gli attuatori; oppure si può fare in modo che il gestore degli attuatori invii delle richieste REST all'emulatore delle lampadine philips Hue per una visualizzazione immediata dello stato dell'attuatore rappresentato da ciascuna lampadina.

Demo – file di configurazione

- Il componente che tiene traccia delle associazioni pulsante-attuatore potrebbe leggere la configurazione da un file di configurazione (oppure potete inizializzare la tabella delle corrispondenze da programma per semplicità, in un metodo di inizializzazione)
- Analogamente sia il componente pannello di controllo che quello di gestione degli attuatori potrebbe leggere i nomi degli switch/attuatori di sua competenza da un file di configurazione (oppure si può definire l'elenco degli switch attuatori di competenza da programma per semplicità)

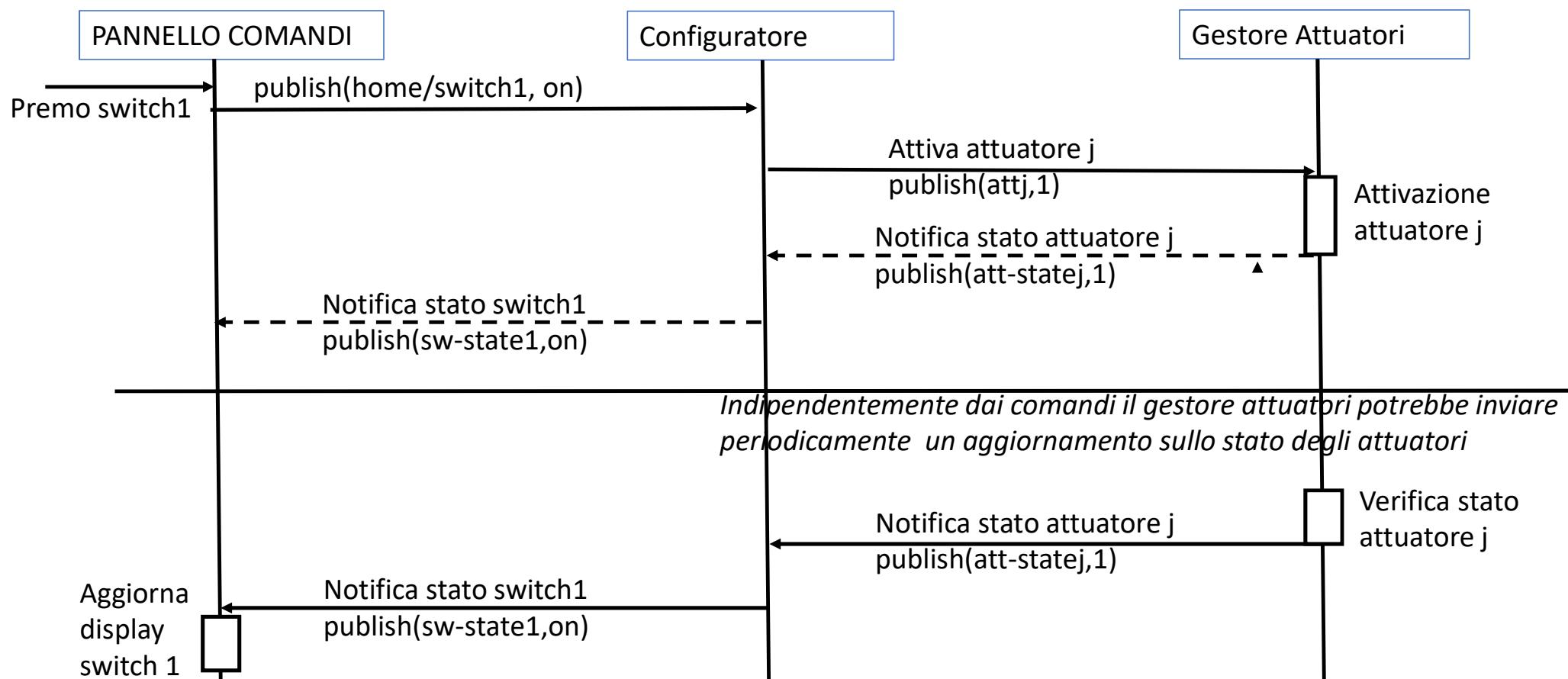
Demo – il Broker

- Inizialmente la sperimentazione utilizzerà il broker su localhost, successivamente si potrà sostituire l'indirizzo con quello di un broker esterno.

Anche se non mostrato esplicitamente c'è sempre un passaggio attraverso il broker (tra pannello comandi e configuratore e tra quest'ultimo e il gestore attuatori)

Interazione tra i componenti

Il configuratore ha una tabella interna che associa ciascuno switch con un determinato attuatore



Mosquitto

controllo degli accessi

Configurazione di mosquito

mosquito ha molte possibilità di configurazione: per modificare i valori di default è possibile agire sul file `mosquitto.conf`

mosquito cerca questo file in una cartella di default, ma è possibile in fase di avvio indicare quale usare
`mosquitto -c mosquito.conf_mia`

Il file di configurazione può contenere degli `#include <dir>`, in questo caso mosquito procede ad esaminare i files inclusi in tali directory come se fosse un unico file che li concatena tutti.

Introduzione di autenticazione utenti

Il file mosquito.conf può essere configurato per limitare gli accessi al broker:

- Permettere la connessione solo ad utenti accreditati
- Limitare i topic accessibili a determinati utenti

Limitazione connessione ad utenti autenticati

mosquitto.conf (sezione Security)

allow_anonymous false

password_file C:\mosquitto\provapass.txt

Il file delle password deve essere preparato: scrivere in un file di testo

utente1:passwU1

utente2:passwU2

Cifrare le password con il comando `mosquitto_passwd -U filepass.txt`

NOTA: dettagli su <https://mosquitto.org/man/mosquitto-conf-5.html>

Attivare mosquitto con autenticazione utenti

```
mosquitto -c mosquitto.conf -v
```

Se proviamo a connettere il client (publisher o subscriber che sia) senza specificare un utente valido viene rifiutata la connessione

```
mosquitto_sub -h localhost -p 1883 -t temperatura
```

ERRORE: RIFIUTATA LA CONNESSIONE

```
mosquitto_sub -h localhost -u user1 -P passwU1 -p 1883 -t temperatura
```

```
mosquitto_pub -h localhost -u user2 -P passwU2 -p 1883 -t temperatura  
-m 23
```

Estendere il programma java visto a lezione

```
String password = "passwU1";
char pwd[] = password.toCharArray();
MqttConnectOptions options = new MqttConnectOptions();
options.setUserName("user1");
options.setPassword(pwd);
```

Controllo sui topic

E' anche possibile limitare l'accesso ai singoli topic definendo una access control list:
in mosquito.conf inserire il nome del file in corrispondenza del parametro acl_file
acl_file C:\mosquitto\acl.txt

```
# user1 può leggere o scrivere qualsiasi topic
user user1
topic readwrite #
# user2 può leggere/scrivere solo topic home/temperatura
user user2 (readwrite si può omettere)
topic home/temperature
#user3 può solo leggere la temperatura
user user3
topic read home/temperature
```

Utilizzare lo username per indicare il topic

È possibile usare lo username all'interno dei topic
permessi

home/%u/att

Se la definizione gerarchica dei topic è fatta in modo da
distinguere le competenze specifiche di un certo user
includendo il suo nome nella composizione del topic
stesso

Naturalmente possiamo usare anche + e # secondo la
stessa logica dell'uso di queste wildcard in fase di
sottoscrizione

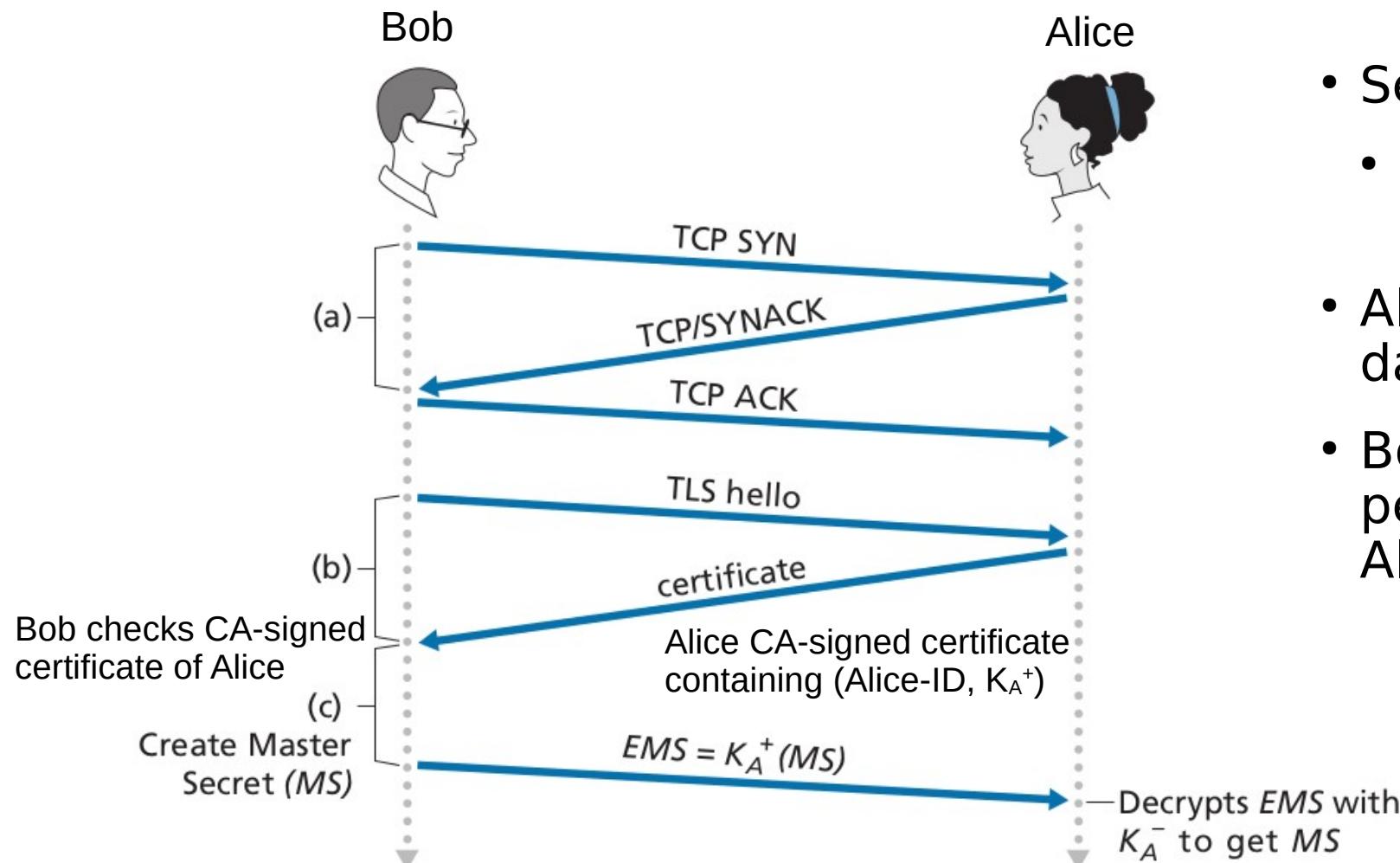
Mosquitto su TLS

Per usare i canali cifrati occorre disporre dei necessari certificati e chiavi ed inoltre bisogna configurare opportunamente mosquitto (file `mosquitto.conf`)

Usiamo openssl per generare certificati e chiavi

Si veda `man mosquitto-tls` per istruzioni dettagliate o
<https://mosquitto.org/man/mosquitto-tls-7.html>

Handshake TLS



- Server-side TSL authentication
 - Il più usato, si autentica solo il server
 - Alice manda certificato firmato da CA
 - Bob usa il certificato della CA per verificare il certificato di Alice

Generare il certificato e le chiavi

- Creare la coppia di chiavi per la nostra CA
- Creare il certificato della CA firmato con la chiave privata della CA
- Creare una coppia di chiavi per il broker
- Creare il certificato del broker e firmarlo con la chiave della CA

Nel file `mosquitto.conf` occorre specificare i pathname dove si trovano i certificati:

`cafile`

`certfile`

`keyfile`

La nostra Certification Authority

È necessario disporre del certificato della CA (la nostra) che ha firmato il certificato del broker.

Inoltre il broker deve avere un suo certificato, garantito dalla CA .

Il **broker** deve anche avere una **chiave privata** per cifrare i propri messaggi

Creazione del certificato e della chiave della CA

```
openssl req -new -x509 -days 600 -extensions v3_req -keyout  
ca.key -out ca.crt -config reqCA.conf --verbose
```

Si devono inserire diversi dati contenuti nel file `reqCA.conf` e una password (inseriamo `pissir`),

Ottengo `ca.crt` e `ca.key`

Contenuto ReqCA.conf

```
[req]
distinguished_name = req_distinguished_name
req_extensions = v3_req
prompt = no

[req_distinguished_name]
C = IT
ST = Vercelli
L = VC
O = UPO
OU = Rettorato # distinguished name of server must not match the distinguished name of CA
CN = localhost

[v3_req]
keyUsage = keyEncipherment, dataEncipherment, keyCertSign # keyCertSign to sign server cert
basicConstraints=CA:TRUE # specify that it is a CA
```

Preparare il certificato per il server

Nel nostro caso la CA è «in casa», comunque il comando è:

```
openssl genrsa -out server.key 2048
```

Ora abbiamo la server.key e dobbiamo farla firmare dalla CA.

```
openssl req -out server.csr -key server.key --new -config  
reqServ.conf --verbose
```

Nuovamente I dati del server sono presenti nel file reqServ.conf.

Attenzione a dare come common name il domain name del server (uso localhost)

A questo punto abbiamo server.csr e server.key e dobbiamo usare la chiave della CA per “firmare” il certificato del server.

Contenuto reqServ.conf

```
[req]
distinguished_name = req_distinguished_name
req_extensions = v3_req
prompt = no

[req_distinguished_name]
C = IT
ST = Vercelli
L = VC
O = UPO
OU = DISIT # distinguished name of server must not match the distinguished name of CA
CN = localhost

[v3_req]
keyUsage = keyEncipherment, dataEncipherment
```

Firmare il certificato del server con la chiave della CA

```
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -  
CAcreateserial -out server.crt -days 500
```

(l'ultimo parametro dice per quanti giorni varrà il certificato).

Output:

Signature ok

subject=C = IT, ST = Vercelli, L = VC, O = UPO, OU = DISIT,
CN = localhost

Getting CA Private Key

Enter pass phrase for ca.key: <<< *inserire la passphrase:* pissir

Ora abbiamo anche **server.crt**

Configurare il broker

In `mosquitto.conf` dobbiamo aggiungere

- 1) Il listener sulla porta 8883
- 2) Dove trovare i file con i certificati impostando i parametri:

`cafile` - certificato della certification authority

`certfile` - certificato del broker

`keyfile` - chiave del broker

Connessione sulla porta 8883

```
./mosquitto_sub -h localhost -p 8883 -t  
pissir/prova --cafile pathname_file_ca.crt
```

```
./mosquitto_pub -h localhost -p 8883 -t  
pissir/prova -m "ciao Ciao" --cafile  
pathname_file_ca.crt
```

L'hostname deve essere identico al Common Name (CN) del certificato, altrimenti la connessione viene rifiutata!

Introduzione all'Internet of Things (IoT)

Cosa è la IoT ?

Il termine IoT è attribuito a Kevin Ashton che l'ha coniato in una sua presentazione del 1999.

<http://www.itrco.jp/libraries/RFIDjournal>

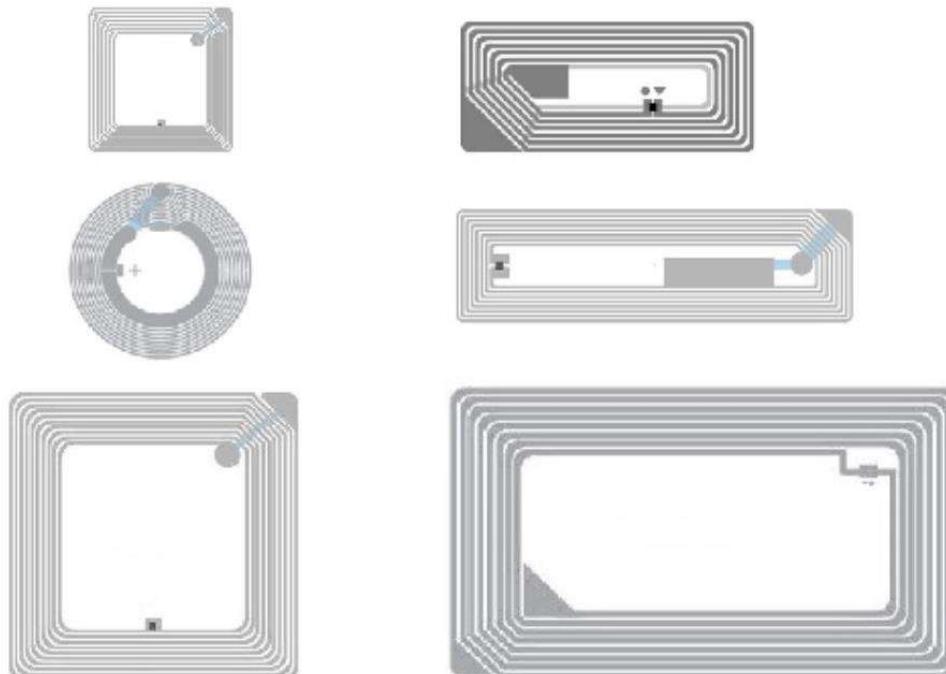


And that's a big deal. We're physical, and so is our environment. Our economy, society and survival aren't based on ideas or information—they're based on things. You can't eat bits, burn them to stay warm or put them in your gas tank. Ideas and information are important, but things matter much more. Yet today's information technology is so dependent on data originated by people that our computers know more about ideas than things.

If we had computers that knew everything there was to know about things—using data they gathered without any help from us—we would be able to track and count everything, and greatly reduce waste, loss and cost. We would know when things needed replacing, repairing or recalling, and whether they were fresh or past their best.

<http://www.itrco.jp/libraries/RFIDjournal-That%20Internet%20of%20Things%20Thing.pdf>

I Tag RFID



Vengono attivati da un lettore quando entrano nel suo raggio d'azione: un micro chip contiene i dati. Non hanno bisogno di alimentazione.
Diverse categorie in base alla frequenza.
LF pochi cm; HF (NFC) 10 cm, leggibili da Smartphone; UHF alcuni metri; SHF 100 m: Tag attivi (es. Telepass)



Definizioni di Internet of Things

"The Internet of Things (IoT) is the network of **physical** objects that contain embedded technology to **communicate** and **sense** or **interact** with their internal states or the external environment."

(Gartner)

"At its core, IoT is simple: it's about connecting devices over the Internet, letting them **talk** to us, applications, and each others."

(The Guardian)

Definizioni di Internet of Things

"The IoT is a giant network of connected 'things' (which also includes people). The relationship will be between **people-people**, **people-things**, and **things-things**."

(Forbes)

"An IoT is a network that connects uniquely identifiable 'Things' to the Internet. The 'Things' have **sensing/actuation** and potential **programmability** capabilities. [...] information about the 'Thing' can be collected and the state of the 'Thing' can be changed from anywhere, anytime, by anything."

(IEEE Internet of Things)*

**Institute of Electrical and Electronics Engineers*

Definizione da un documento della IEEE

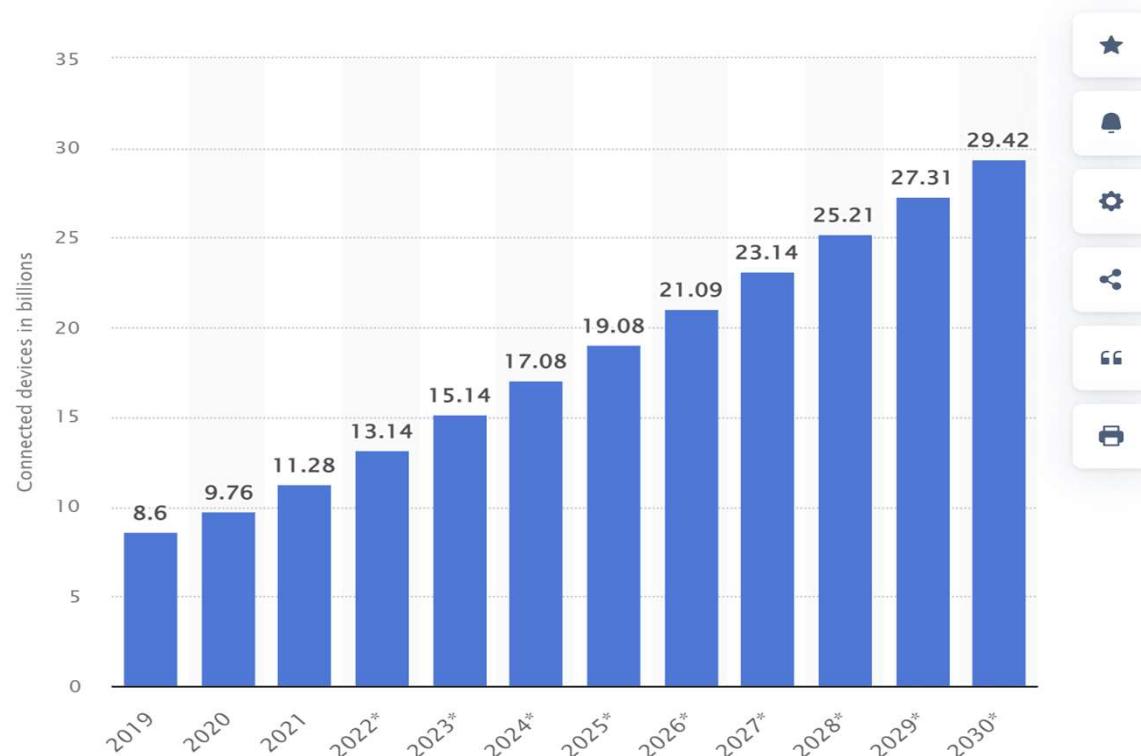
http://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf

- An IoT system is a system that deals with the interconnection of any physical object that is relevant from a **user or application perspective**.
- Sensors/actuators are connected to the “Things” and perform the sensing/actuation which bring the *smartness* of the “Things.”
- Has communication capabilities based on interoperable protocols and have self-configuration capabilities
- The “Things” of an IoT system have a programmability feature (may change behavior at a user’s command)

(IEEE Internet of Things)

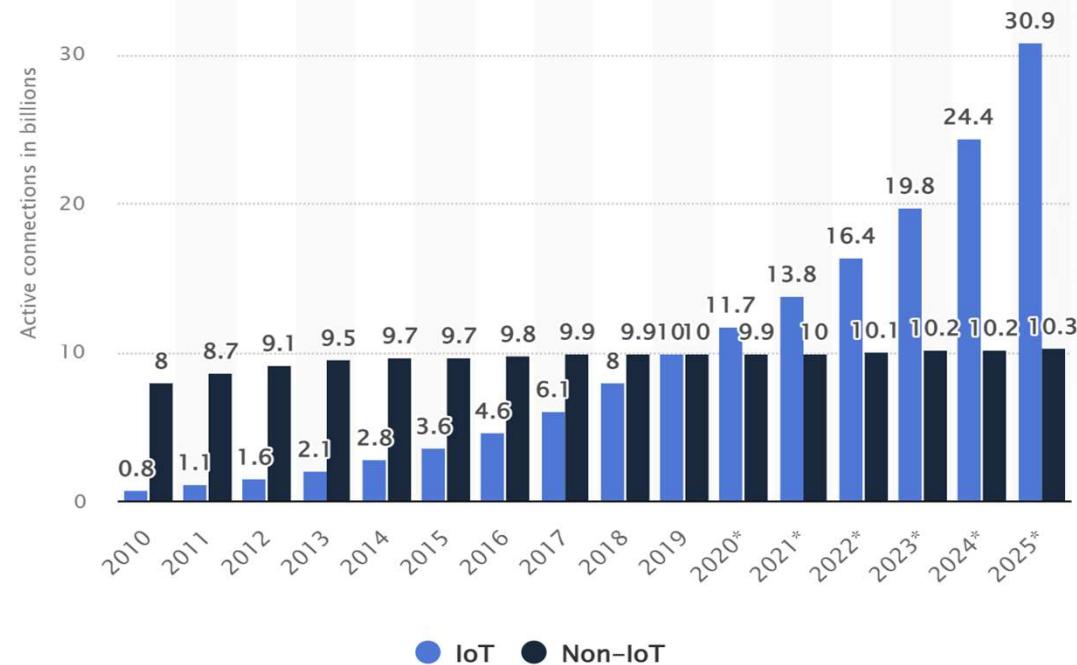
Il numero di oggetti connessi sta crescendo rapidamente

<https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/>

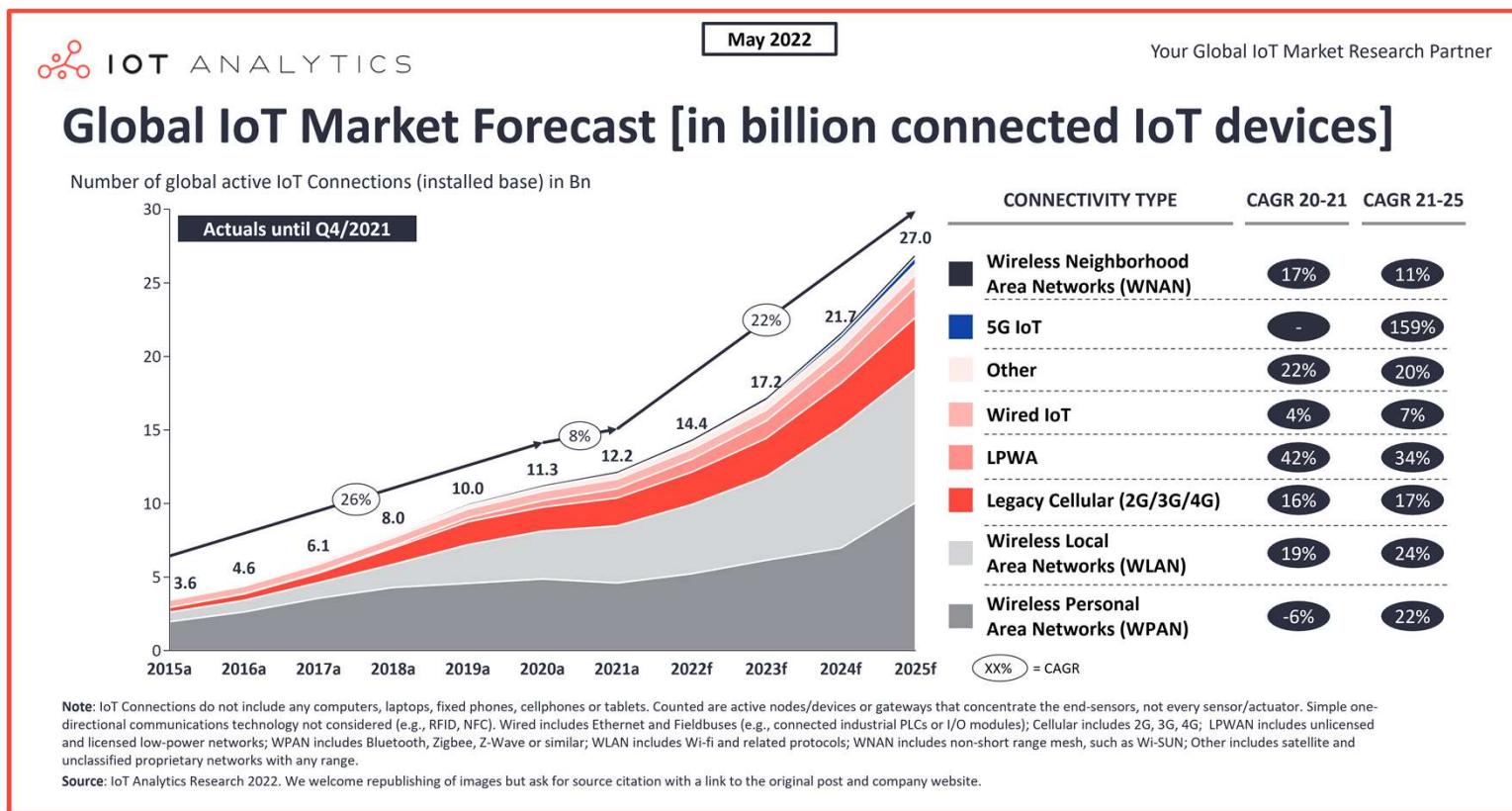


... e crescerà di più del numero di computer connessi

<https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/>



Il mondo IoT è un ambito di grandi investimenti



Le applicazioni: Smart *

- Smart home (domotica, ma anche intrattenimento)
- Smart mobility (inclusi i veicoli a guida autonoma, il car sharing, i servizi di monitoraggio real time dei mezzi pubblici, ...)
- Smart factory (Industria 4.0)
- Smart logistics
- Smart grid
- Smart city (inclusi aspetti di Sicurezza)
- Assistenza sanitaria continua (Smart health?)

Smart Grids (qui si fa riferimento anche all'uso di 5G)

5G IoT Use cases #3: Smart Grid Automation

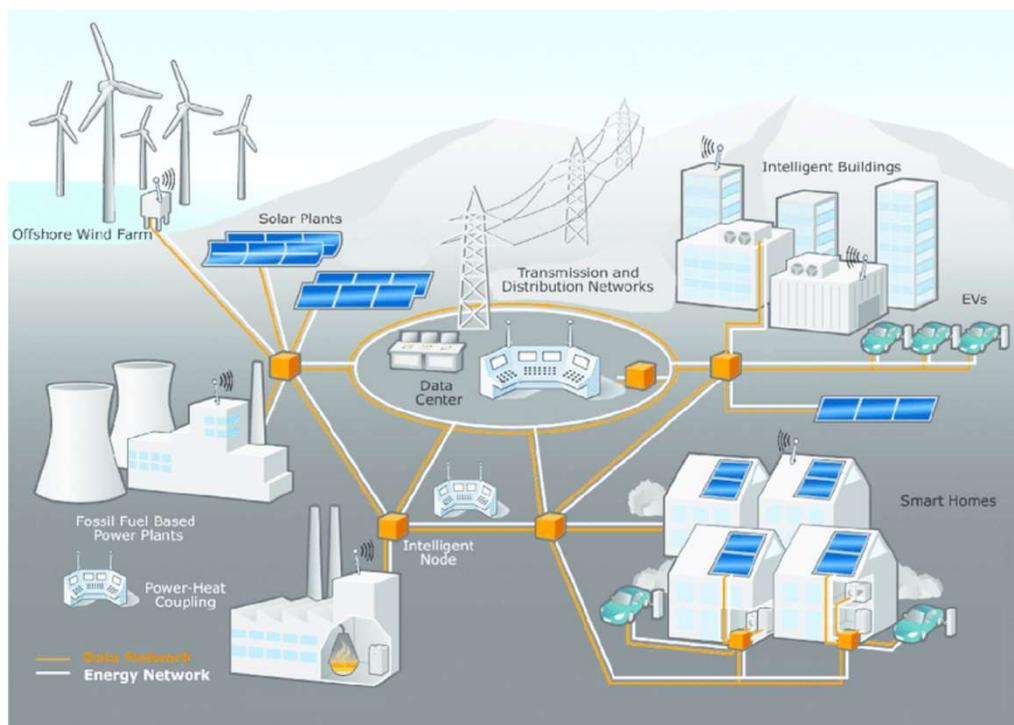
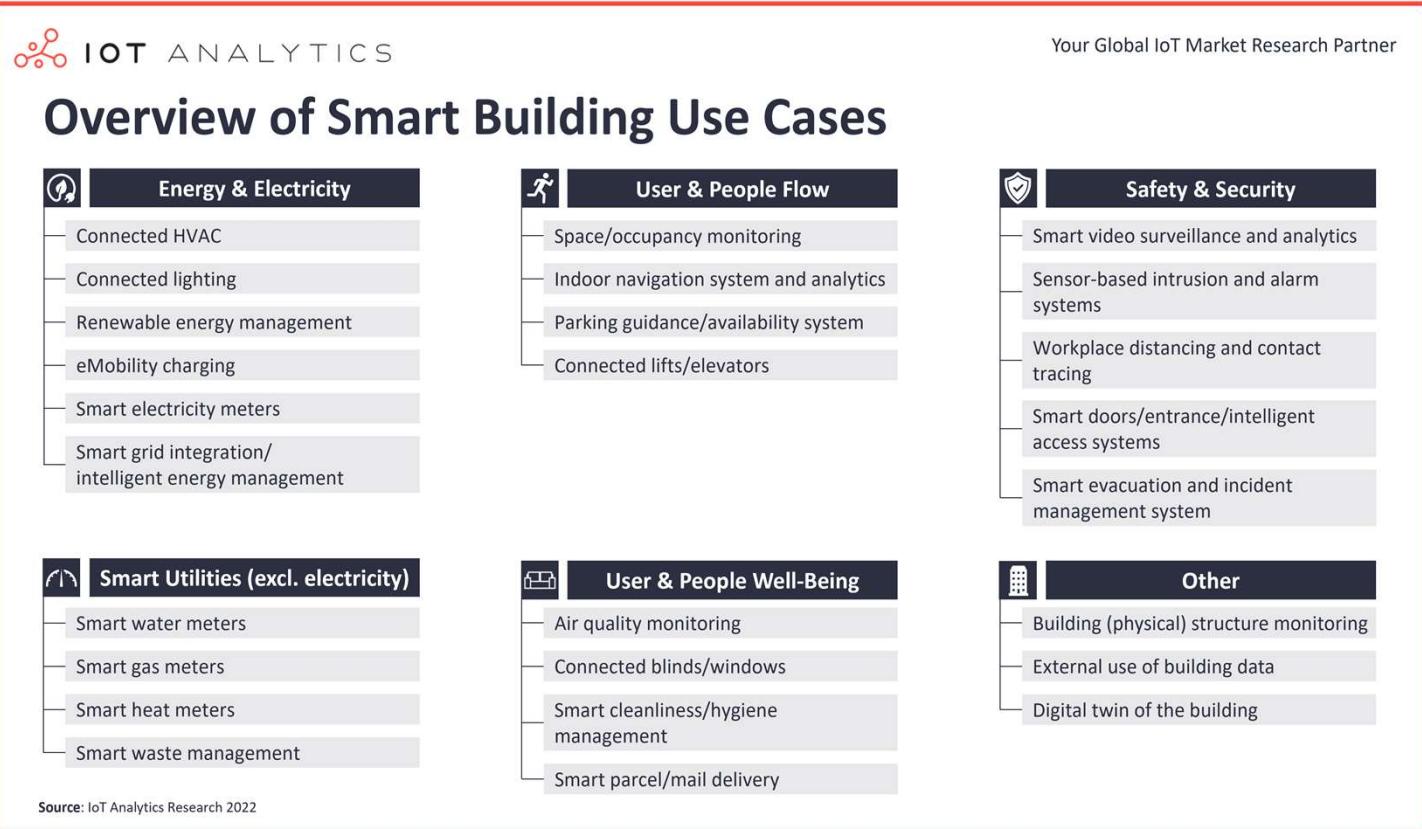


Exhibit 4: Example of a smart grid and its communication network. Image credit: ABB, Deutsche Telekom.

Un esempio di ambito applicativo: smart buildings

<https://iot-analytics.com/our-coverage/smart-buildings/>



The screenshot shows a section of the IoT Analytics website titled "Overview of Smart Building Use Cases". The page features a red border and includes the IoT Analytics logo and tagline "Your Global IoT Market Research Partner". It is organized into six categories, each with an icon and a list of use cases:

- Energy & Electricity** (Icon: Power plug):
 - Connected HVAC
 - Connected lighting
 - Renewable energy management
 - eMobility charging
 - Smart electricity meters
 - Smart grid integration/intelligent energy management
- User & People Flow** (Icon: Person walking):
 - Space/occupancy monitoring
 - Indoor navigation system and analytics
 - Parking guidance/availability system
 - Connected lifts/elevators
- Safety & Security** (Icon: Shield):
 - Smart video surveillance and analytics
 - Sensor-based intrusion and alarm systems
 - Workplace distancing and contact tracing
 - Smart doors/entrance/intelligent access systems
 - Smart evacuation and incident management system
- Smart Utilities (excl. electricity)** (Icon: Water drop):
 - Smart water meters
 - Smart gas meters
 - Smart heat meters
 - Smart waste management
- User & People Well-Being** (Icon: Stethoscope):
 - Air quality monitoring
 - Connected blinds/windows
 - Smart cleanliness/hygiene management
 - Smart parcel/mail delivery
- Other** (Icon: Building):
 - Building (physical) structure monitoring
 - External use of building data
 - Digital twin of the building

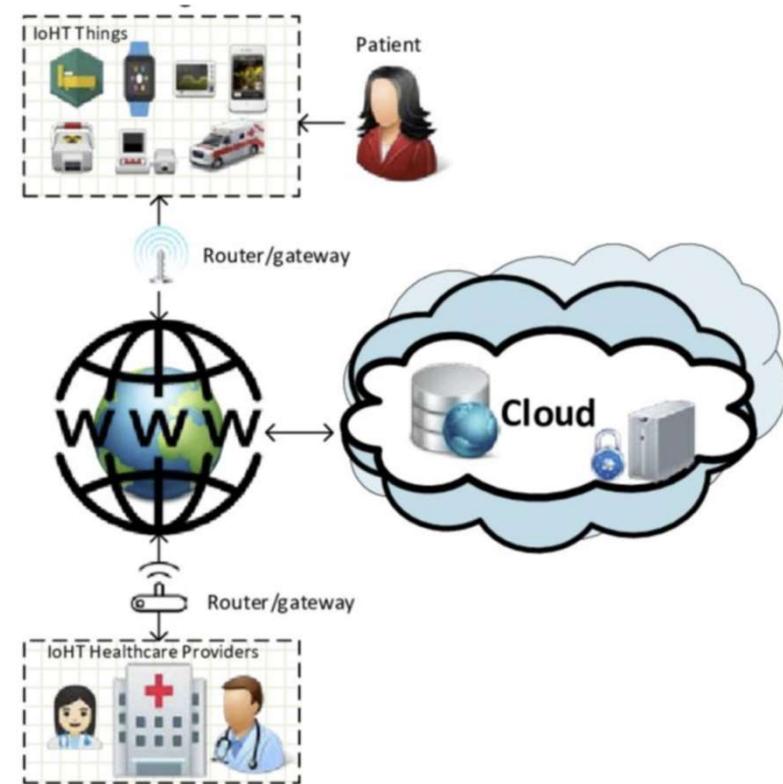
Source: IoT Analytics Research 2022

Alcuni dispositivi che si possono connettere in rete ...



Oggi si parla anche di Internet of Health Things (IoHT)

Tramite un «gateway» e grazie alla rete internet i dati relativi ai parametri vitali di un paziente possono essere raccolti in un deposito in rete, letti da personale medico, incrociato con la cartella clinica del paziente ed utilizzato per decidere se intervenire in suo soccorso o fornire al paziente indicazioni terapeutiche.



... e di IoE(Internet of Everything)

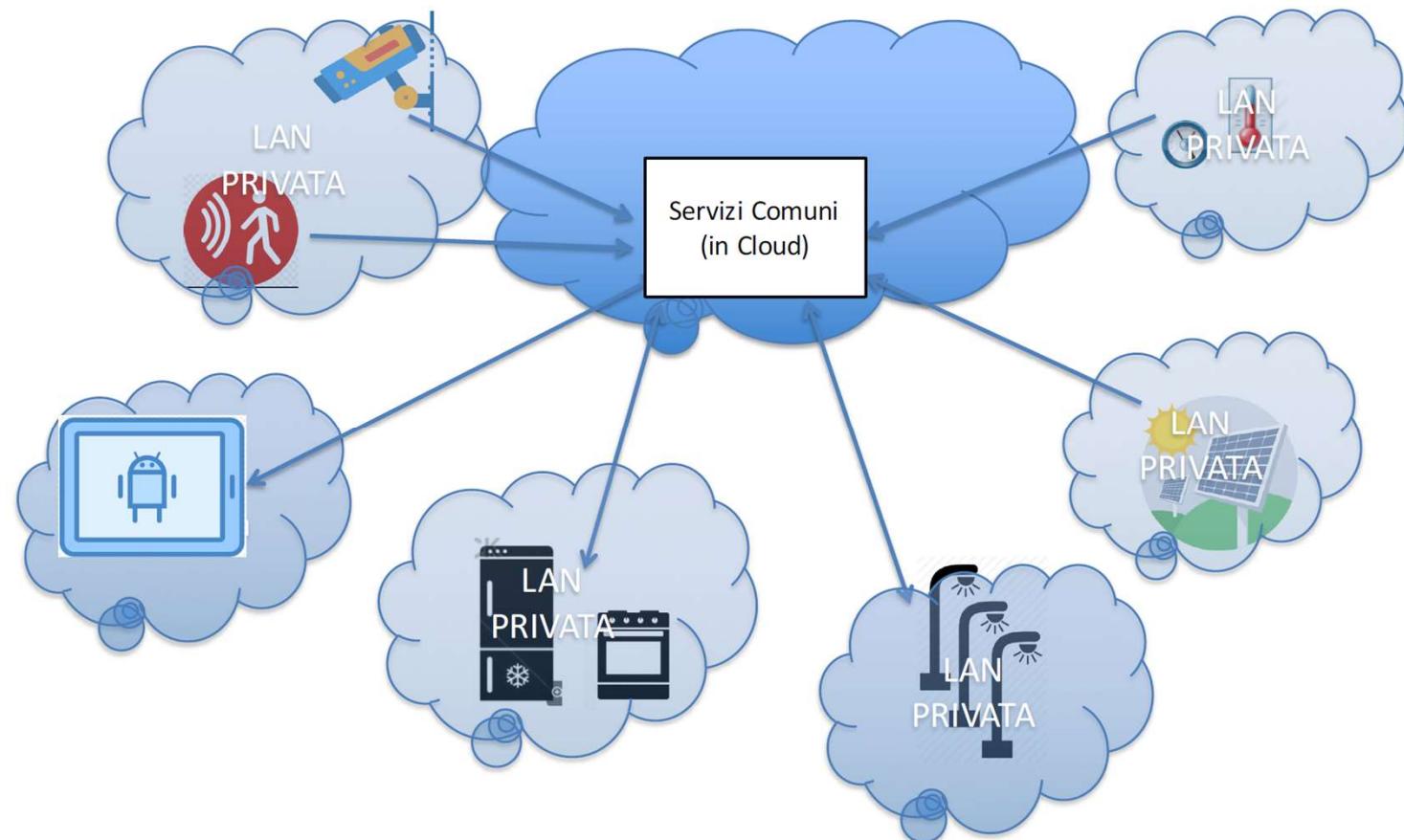
E' una estensione dell'IoT che prevede di collegare nella rete globale:

- Cose (IoT)
- Persone (Sempre connesse tramite smartphone e oggetti indossabili)
- Dati (Big Data raccolti dalla moltitudine di «sensori» connessi)
- Processi (Analizzando i dati raccolti si possono ottimizzare per esempio i processi organizzativi aziendali)

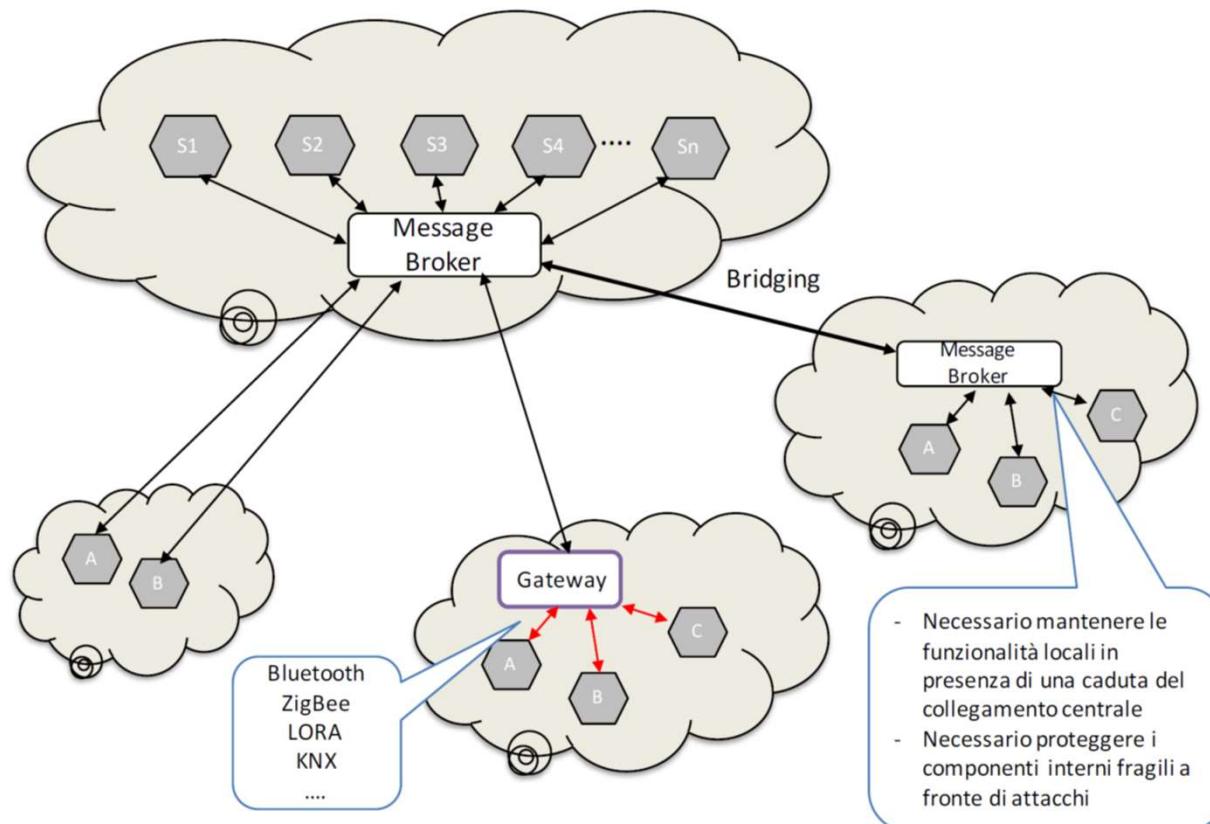
«L'IoE è destinato a portare a compimento un processo di fusione tra il mondo fisico e quello virtuale»

physical + digital = phygital

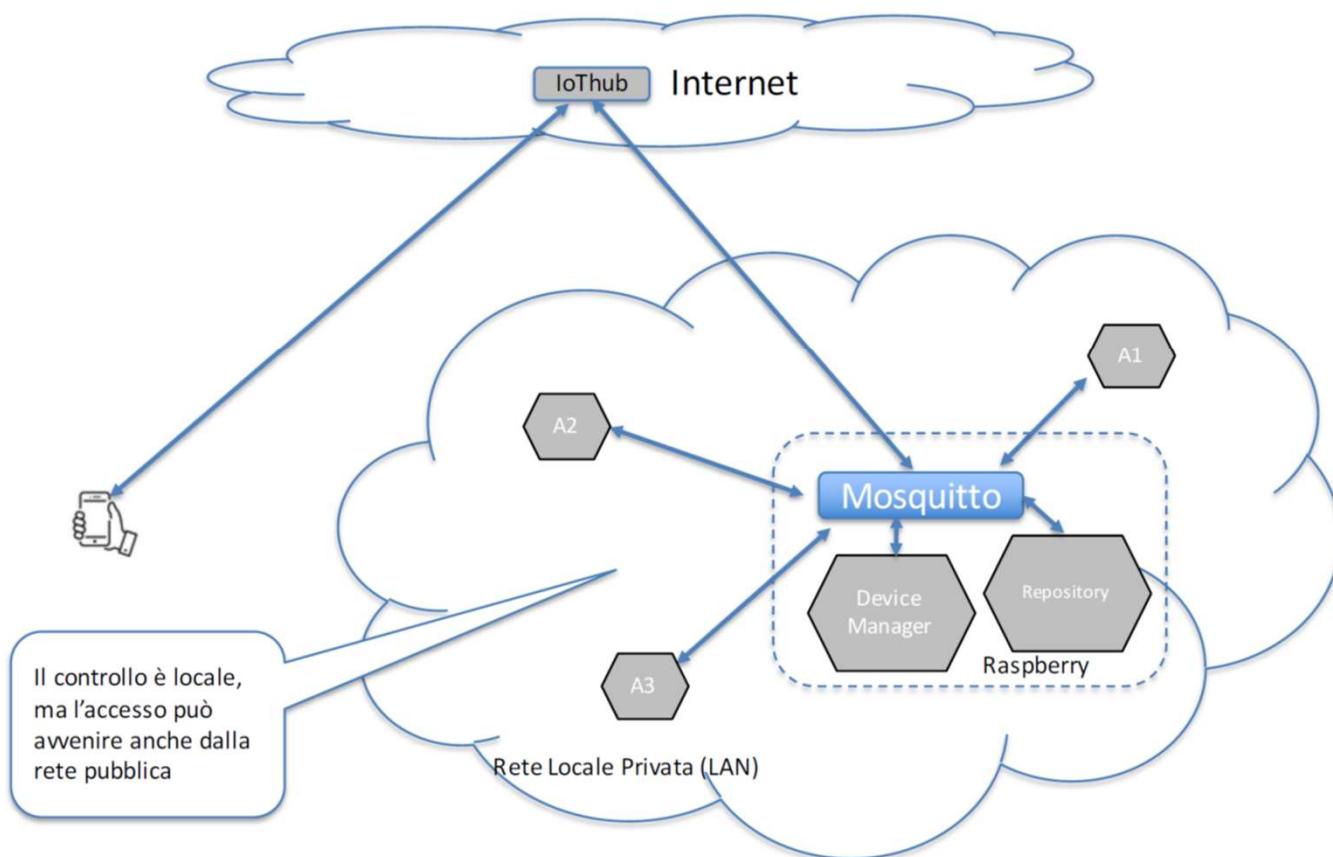
Uno scenario di sistema IoT distribuito geograficamente



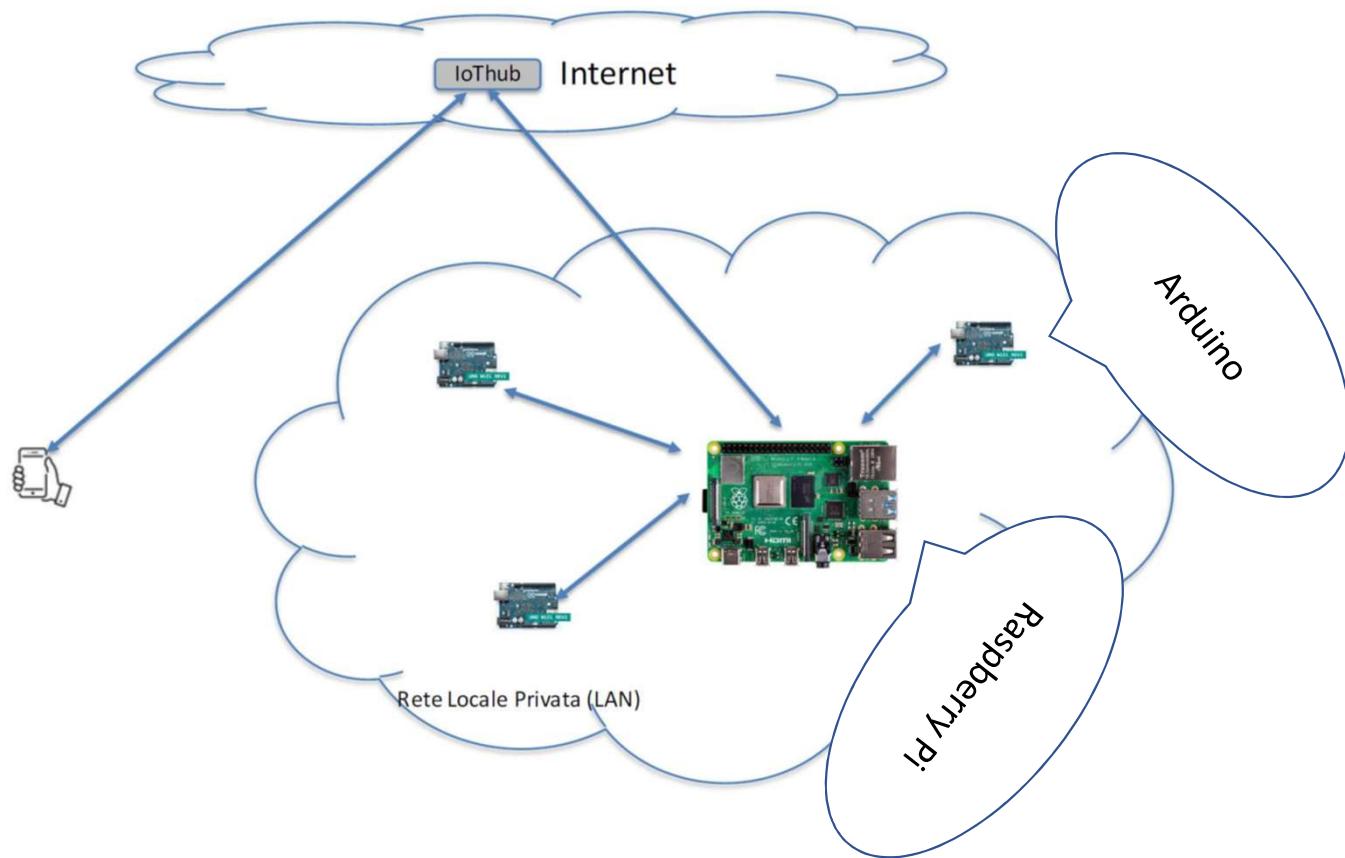
Tipica Architettura IoT



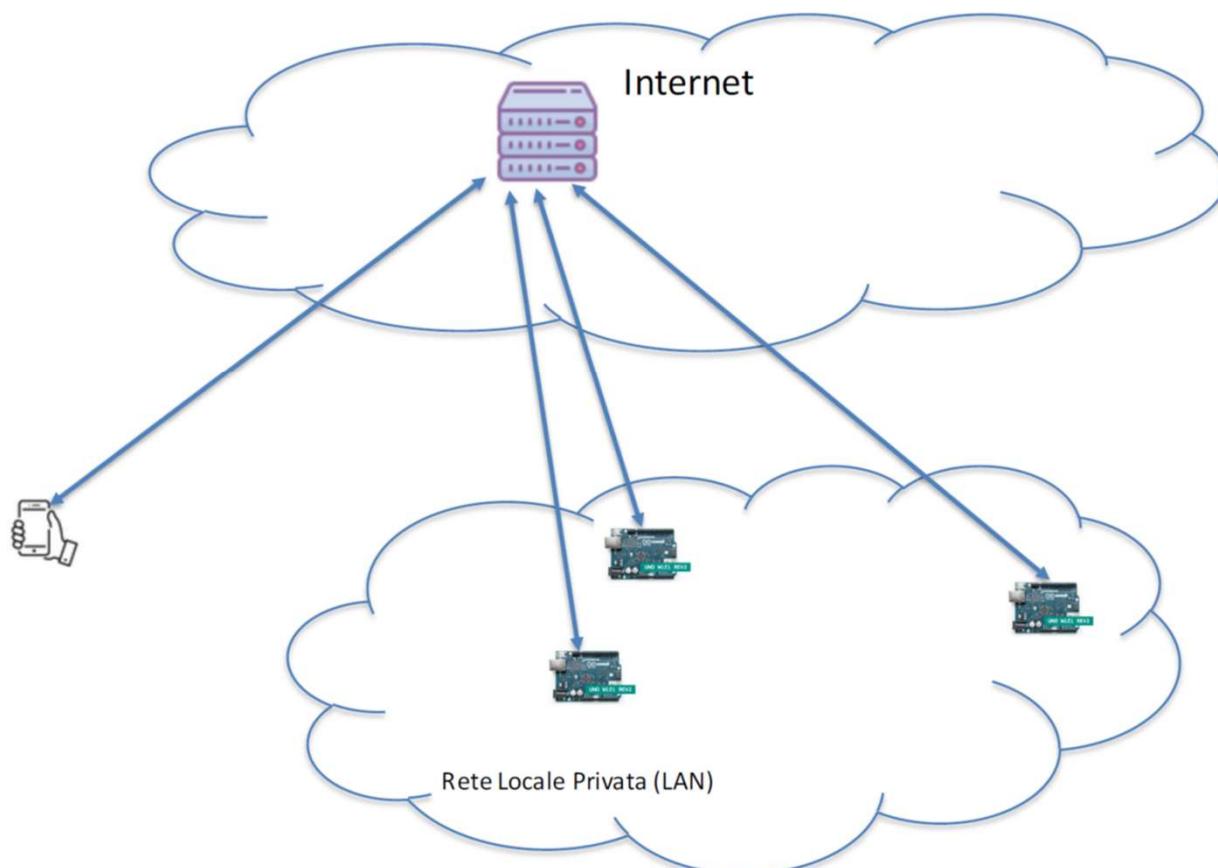
Architetture Basate su LAN



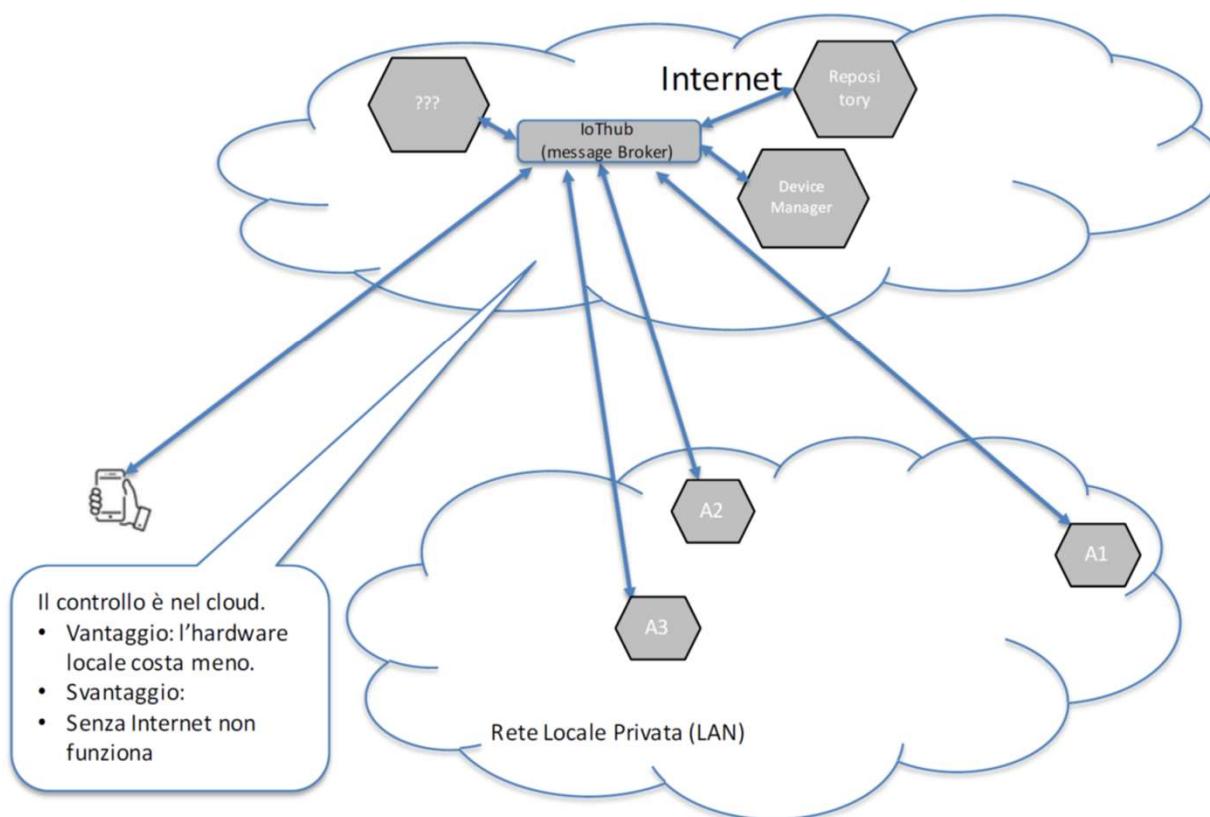
Architetture Basate su LAN



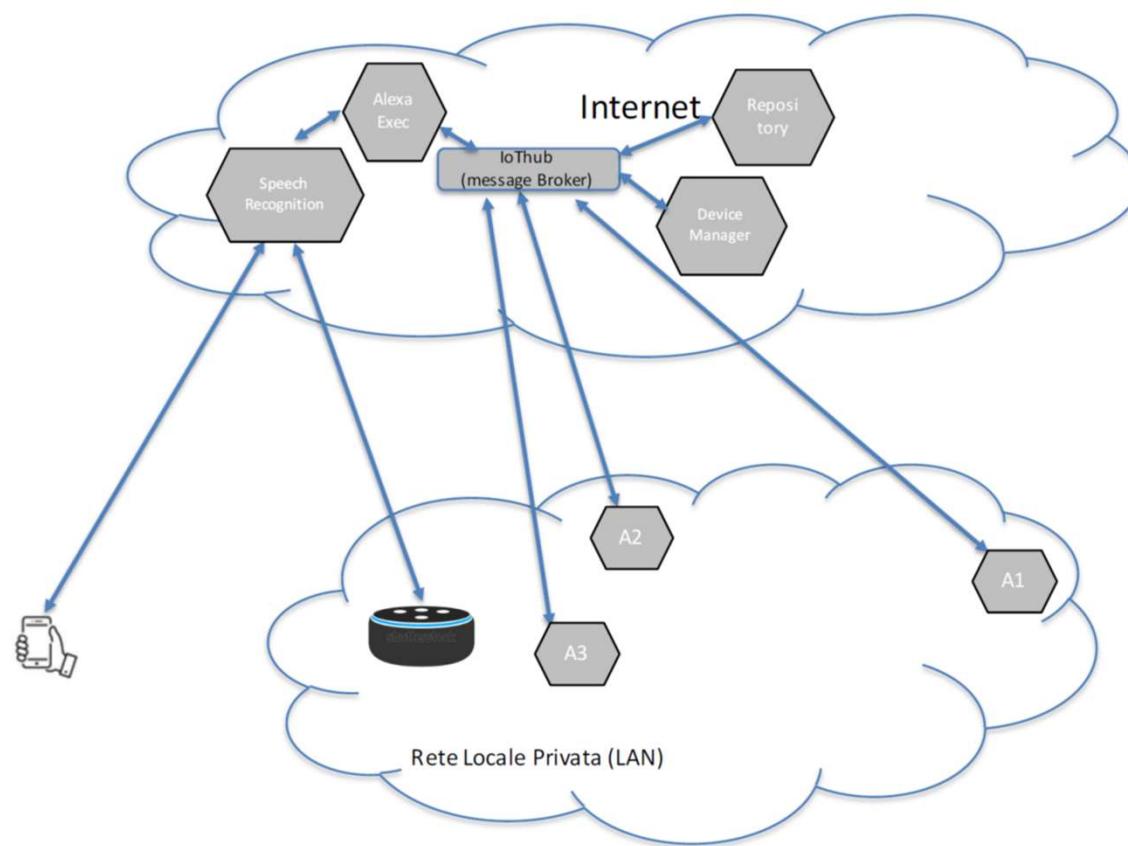
Architetture Basate su Cloud



Architetture Basate su Cloud



Architetture IoT Basate su Controllo Vocale



Microservizi

In questo contesto un'architettura a microservizi risulta particolarmente adatta per la sua modularità: i diversi servizi possono essere posizionati in diversi punti della rete e connessi tramite broker, usando il protocollo MQTT o altri protocolli di scambio di messaggi come per esempio AMQP (Advanced Message Queueing Protocol).

Philips Hue

Esempio di interazione con lampadine Philips Hue

Sistema di home automation: Bridge + lampadine/striscia LED



Il bridge è collegato alla LAN di casa, ha un suo indirizzo IP, ed espone una interfaccia REST

Il bridge forma una PAN con lampadine o strisce LED utilizzando il **protocollo Zigbee** (profilo Smart Lighting), un **protocollo wireless a basso consumo, aperto**.



Bridge e lampadine

- Le chiamate alle [API REST](#) del bridge possono essere usate per leggere lo stato delle lampadine, o per modificarlo. Il bridge comunicherà con la lampadina via Zigbee per impostare lo stato.
- Si può accedere al bridge tramite un'app della Philips (per potersi collegare occorre essere fisicamente vicino al bridge perché per associare il cellulare al bridge occorre premere un pulsante sul bridge stesso) oppure si può sviluppare un client che effettua le chiamate RESTful al bridge. Tipicamente il bridge non si espone su internet, ma viene visto solo nella intranet.

TUTTE LE API DEL BRIDGE SI POSSONO TROVARE QUI (Nota: occorre registrarsi):

<https://developers.meethue.com/develop/hue-api/>

L'emulatore

Esiste un emulatore che implementa (parte del)le funzionalità disponibili anche sul sistema reale. Per scaricare il jar:

<http://steveyo.github.io/Hue-Emulator/>

- Una volta scaricato il jar dell'emulatore lo si può avviare
- Dalla GUI scegliere la porta su cui offre il servizio e premere Start
(Nota: si può verificare con CURL che risponda alle richieste)

```
curl localhost:8000/api/newdeveloper/lights
```

RESTful API

URL per accedere al server (emulatore o bridge)

`http://<IP SERVER>:8000/api/<USERNAME>/lights`

`http://<IP SERVER>:8000/api/<USERNAME>/lights/<N>`

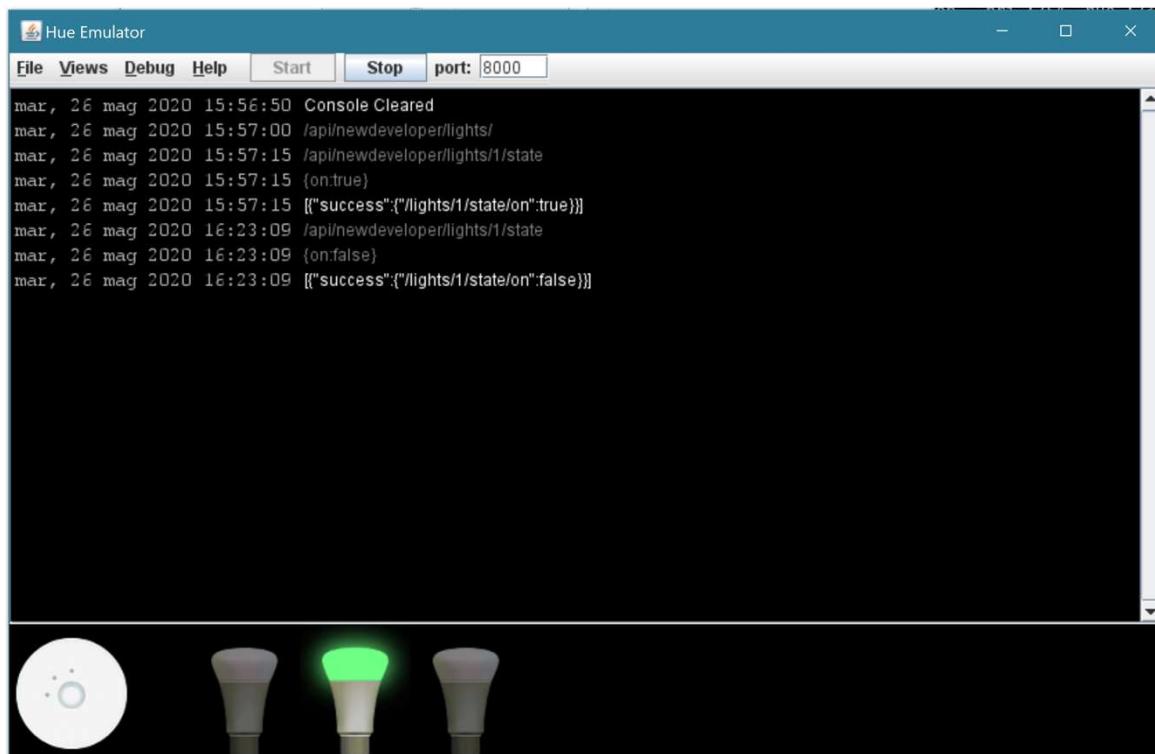
`http://<IP SERVER>:8000/api/<USERNAME>/lights/<N>/state`

Dove:

`<IP SERVER>` è l'indirizzo IP dell'emulatore (localhost) o del bridge

`<USERNAME>` è "newdeveloper" per l'emulatore mentre per il bridge è una stringa lunga e complicata che si può generare tramite la procedura spiegata nella Sezione «Getting Started» del sito per Hue Developer

Emulatore per test RESTful API



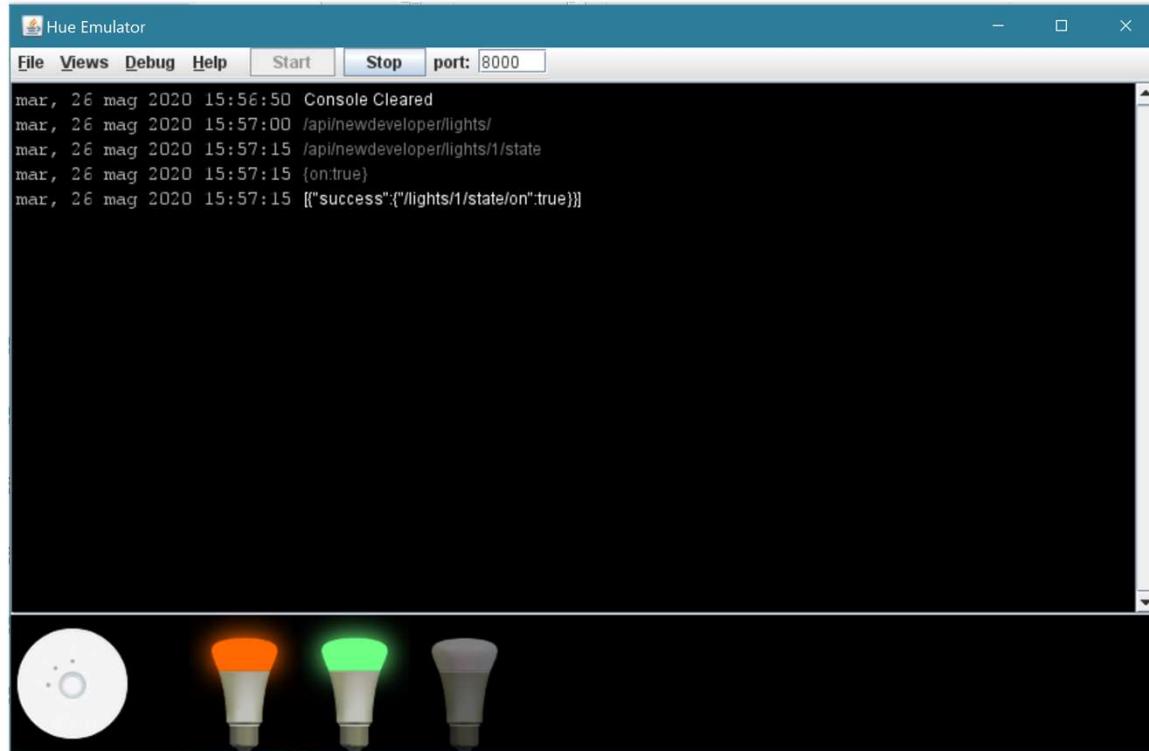
curl <http://localhost:8000/api/newdeveloper/lights/>

Legge (GET) l'elenco delle lampadine presenti e di ciascuna descrive le caratteristiche tecniche e lo stato.

VALORE RESTITUITO DALLA GET

```
{"1": {"modelid": "LCT001", "name": "Hue Lamp 1", "swversion": "65003148", "state": {"xy": [0, 0], "ct": 0, "alert": "none", "sat": 254, "effect": "none", "bri": 254, "hue": 4444, "colormode": "hs", "reachable": true, "on": false}, "type": "Extended color light"}, "pointsymbol": {"1": "none", "2": "none", "3": "none", "4": "none", "5": "none", "6": "none", "7": "none", "8": "none"}, "uniqueid": "00:17:88:01:00:d4:12:08-0a"}, {"2": {"modelid": "LCT001", "name": "Hue Lamp 2", "swversion": "65003148", "state": {"xy": [0.346, 0.3568], "ct": 201, "alert": "none", "sat": 144, "effect": "none", "bri": 254, "hue": 23536, "colormode": "hs", "reachable": true, "on": true}, "type": "Extended color light"}, ... }
```

Accendere una lampadina



```
$ curl -X PUT -d "{on:true}"
http://localhost:8000/api/newdeveloper/lights/1/state
>>> RISPOSTA [{"success": "/lights/1/state/on": true}])
Vedere: https://developers.meethue.com/develop/hue-api/lights-api/#set-light-state
```

Impostare i colori: HSL

Nello stato delle lampadine si possono impostare i tre parametri:

hue (0:65535)

sat (0:254)

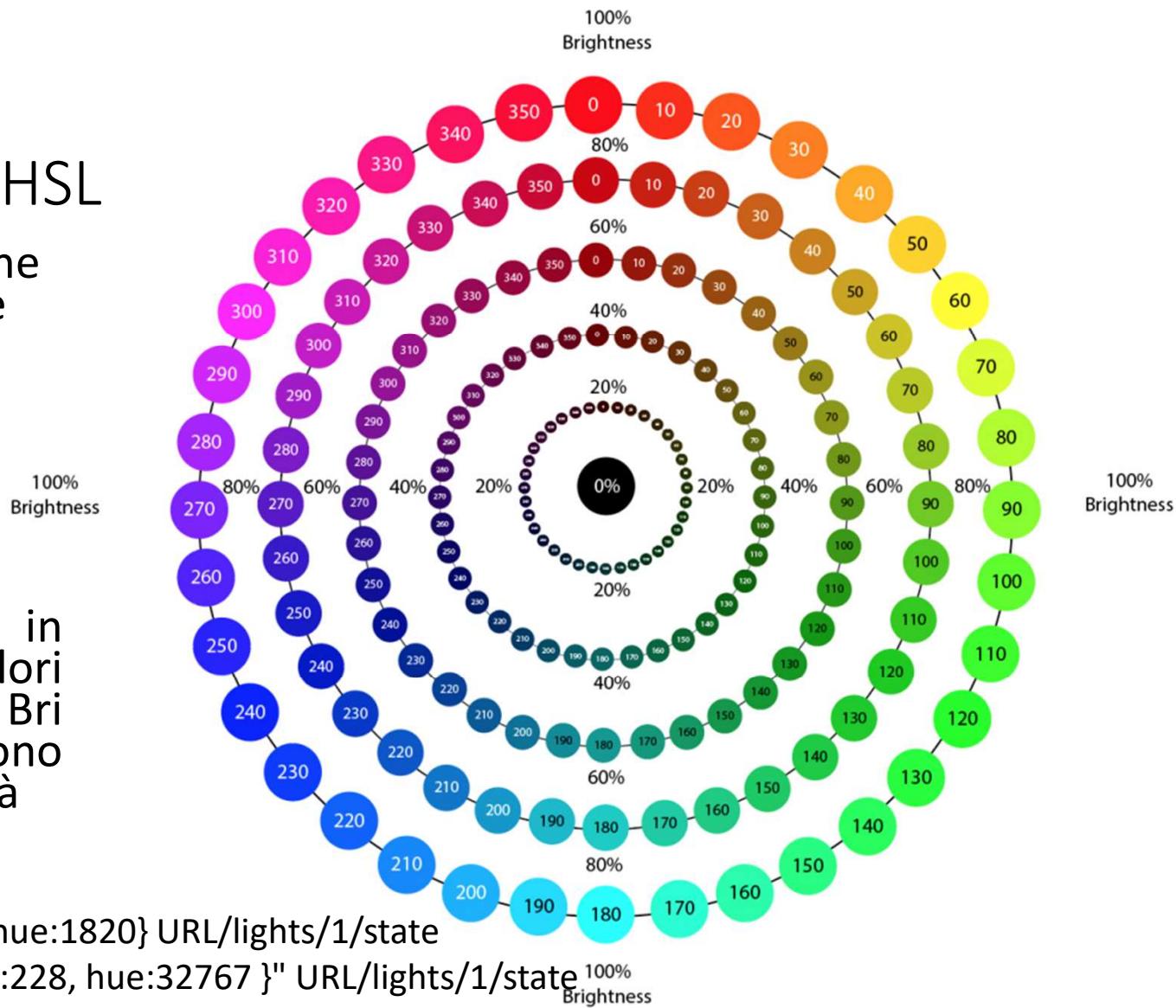
bri (1:254)

Facendo le proporzioni in base alla ruota dei colori (Hue in gradi 0:360; Sat e Bri in percentuale) si possono ottenere le diversi tonalità

Esempi:

ROSA curl -X PUT -d "{bri:228, sat:80, hue:1820}" URL/lights/1/state

TURCHESE curl -X PUT -d "{bri:152, sat:228, hue:32767 }" URL/lights/1/state



Un semplice client che interagisce con l'emulatore

- Vediamo ora un semplice client che interagisce con l'emulatore delle Philips Hue. Se utilizzato in laboratorio può funzionare con le lampadine fisiche semplicemente sostituendo l'URL con l'indirizzo IP del bridge e lo <username> con una stringa ottenuta direttamente dal Bridge con una opportuna procedura da effettuare in loco.

Spring-web framework + GSON

- Per questo esempio utilizziamo una parte del framework Spring che permette di fare chiamate REST (classe RestTemplate). Quest'ultimo utilizza GSON come libreria per gestire la serializzazione e deserializzazione di oggetti Java in JSON.

INSERIRE IN build.gradle:

```
implementation 'org.springframework:spring-web:5.1.5.RELEASE'  
implementation 'com.google.code.gson:gson:2.8.5'
```

La classe EsperimentiHue

- La classe Hue contiene il main() e utilizza le seguenti classi e metodi della libreria spring-web:
 - Classe RestTemplate con i metodi getForObject, put
 - Classe HttpHeaders con il metodo getContentType
 - Classe HttpEntity

Si veda anche: <https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html>

Import

```
import java.util.HashMap;  
import java.util.Map;  
  
import org.springframework.http.HttpEntity;  
import org.springframework.http.HttpHeaders;  
import org.springframework.http.MediaType;  
import org.springframework.web.client.RestTemplate;
```

Un client per comandare le lampadine Philips Hue

```
String baseURL = "http://localhost:8000"; // Oppure IP del Bridge  
String username = "newdeveloper"; // Oppure stringa ottenuta dal Bridge  
String lightsURL = baseURL + "/api/" + username + "/lights/";
```

```
RestTemplate rest = new RestTemplate();
```

La classe RestTemplate implementa l'interfaccia RestOperations (specifica un insieme di operazioni RESTful)

<https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html>

Metodi per le operazioni GET e PUT

```
Map<String, ?> allLights = rest.getForObject(lightsURL, HashMap.class);
```

Il metodo `getForObject(URI url, Class<T> responseType)` fa una GET sulla URL specificata, nel nostro caso ottiene la descrizione completa di tutte le lampadine.

Il server risponde in JSON: le informazioni in esso contenute vengono restituite dal metodo sotto forma di una mappa con:

chiave <id lampadina> e valore <descrizione lampadina>

La descrizione della lampadina a sua volta è articolata in una gerarchia.

Metodi per le operazioni GET e PUT

```
if (allLights != null) { // accendiamo tutte le lampadine impostando l'effetto colorloop  
    HttpHeaders headers = new HttpHeaders();  
    headers.setContentType(MediaType.APPLICATION_JSON);  
    String colorloop = "{\"on\":true, \"effect\": \"colorloop\" }";  
    HttpEntity<String> request = new HttpEntity<>(colorloop,headers);
```

La classe `HttpHeaders` rappresenta l'header di una richiesta/risposta http

Il metodo `setContentType` specifica come è rappresentato il body ([APPLICATION_JSON](#))

La classe `HttpEntity` rappresenta una richiesta/risposta http e consiste di header e body.
Al costruttore passiamo il body (stringa `colorloop`) e l'header (`headers`).

Nota: l'effetto colorloop si vede sulle lampadine vere, non sull'emulatore. Provate a costruire un body diverso per impostare il colore anziché l'effetto.

Metodi per le operazioni GET e PUT

```
for (String light : allLights.keySet()) { // per ogni lampadina nella mappa  
    // (la chiave è l'id della lampadina)  
    { String callURL = lightsURL+light + "/state";  
        rest.put(callURL,request); }  
    }
```

Il metodo put genera una richiesta di tipo PUT sullo stato della lampadina light con richiesta di impostare l'attributo "effect" a "colorloop"

Metodi per le operazioni GET e PUT

Inserire un ritardo per attendere qualche secondo prima di spegnere

```
String off = "{\"on\":false }";  
HttpEntity<String> requestOff = new HttpEntity<>(off,header)  
for (String l : allLights.keySet())  
    { // modificare lo stato di tutte le lampadine "on": false  
        String callURL = lightsURL+l + "/state";  
        rest.put(callURL,requestOff);  
    }
```

Create il progetto in IntelliJ, inserite le dipendenze, copiate il codice poi eseguite avendo prima attivato l'emulatore.

```
public class EsperimentiHue {
    public static void main(String[] args) {
        String baseURL = "http://localhost:8000";
        String username = "newdeveloper";
        String lightsURL = baseURL + "/api/" + username + "/lights/";
        RestTemplate rest = new RestTemplate();
        Map<String, ?> allLights = rest.getForObject(lightsURL, HashMap.class);
        if (allLights != null) {
            HttpHeaders headers = new HttpHeaders();
            headers.setContentType(MediaType.APPLICATION_JSON);
            String colorloop = "{\"on\":true, \"effect\": \"colorloop\" }";
            HttpEntity<String> request = new HttpEntity<>(colorloop,headers);
            for (String light : allLights.keySet())
                {   String callURL = lightsURL+light + "/state";
                    rest.put(callURL,request);
                }

            for (int i=0;i<10;i++)
                { // ATTENDE 10 SECONDI PRIMA DI SPEGNERE
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) { e.printStackTrace(); }
                    System.out.println(10-i);
                }
            String off = "{\"on\":false }";
            HttpEntity<String> requestOff = new HttpEntity<>(off,headers);
            for (String l : allLights.keySet()) {
                String callURL = lightsURL+l + "/state";
                rest.put(callURL,requestOff);
            }
        }
    }
}
```

Esempio creato da Luigi De Russis

ESERCIZIO

- Modificate l'esempio in modo che quando viene accesa ciascuna lampadina il suo colore gradualmente si modifichi (impostando l'attributo hue ad un valore che di volta in volta viene incrementato di 1000); dopo la prima fase le lampadine dovranno avere colori diversi
- Alla fine, quando le lampadine devono essere spente, inserire un ciclo decrementando l'attributo bri (inizialmente impostato al valore massimo) ad ogni iterazione e facendo il put del nuovo valore.

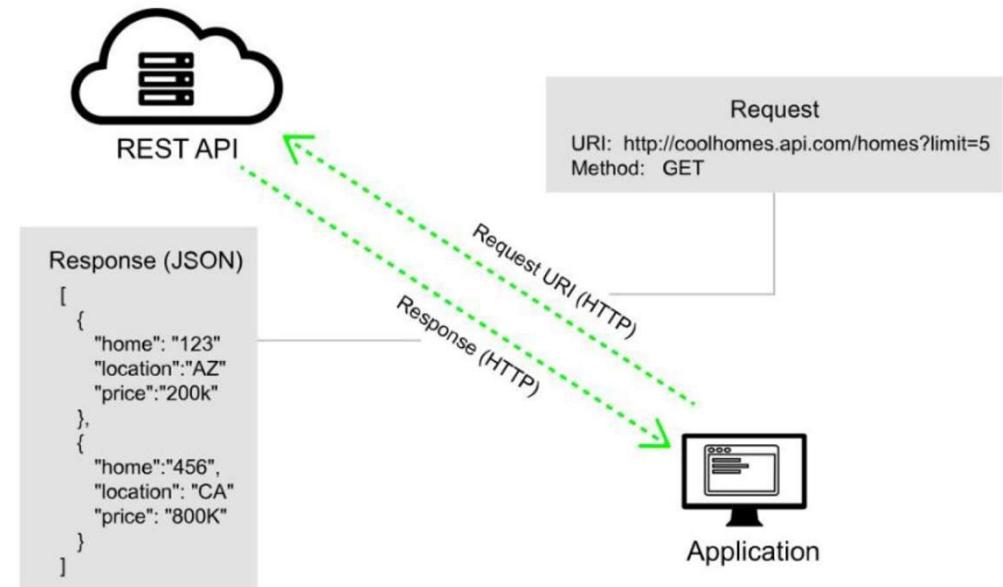


REpresentational State Transfer

Stile architetturale per applicazioni distribuite.

Interazione di tipo client-server veicolato tramite HTTP

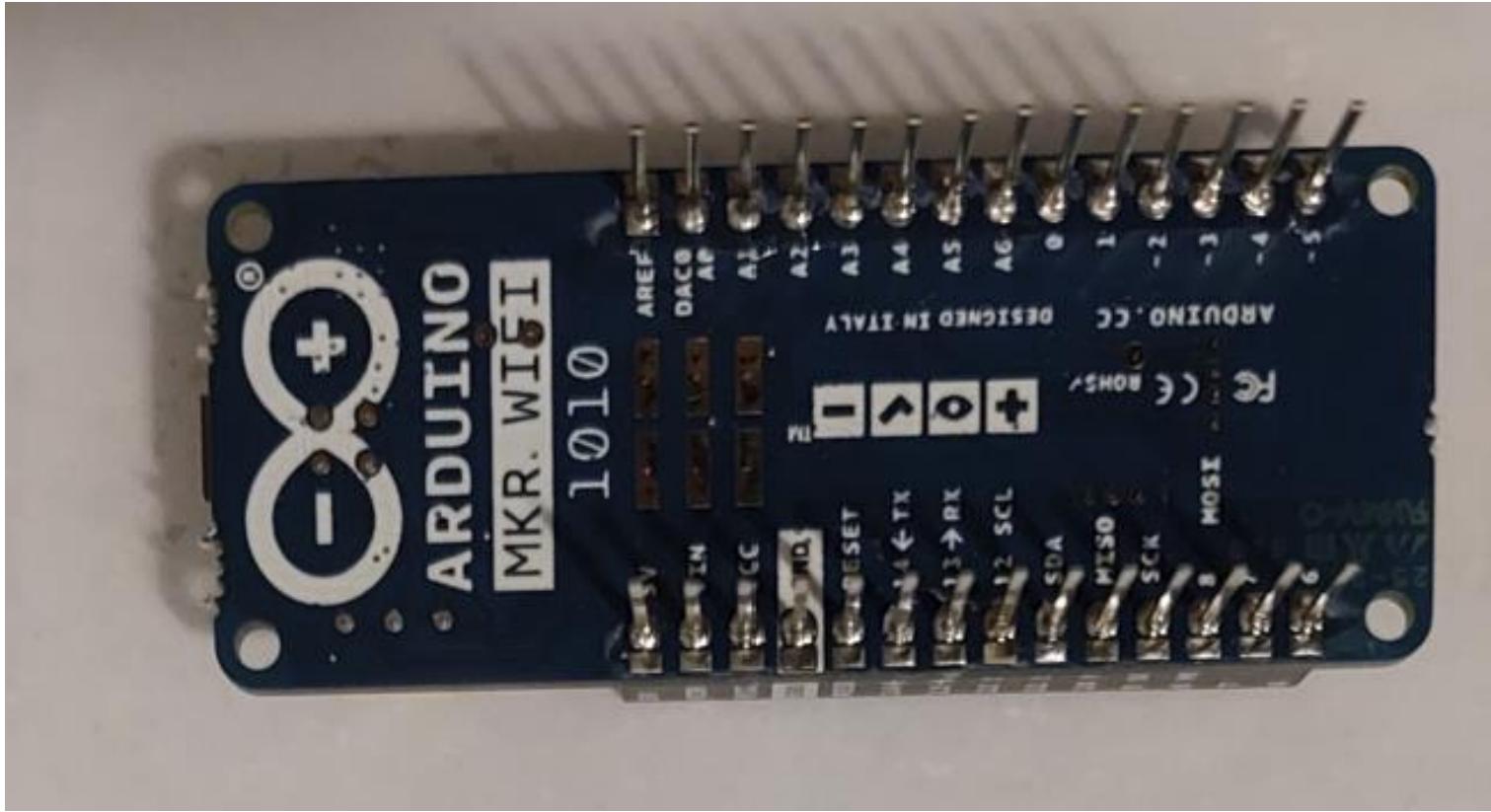
Il server espone delle risorse identificate tramite URI; client e server si scambiano una rappresentazione della risorsa



Le operazioni CRUD si realizzano utilizzando i metodi POST, GET, PUT e DELETE di HTTP

Arduino IoT kit

Arduino MrkWifi1010



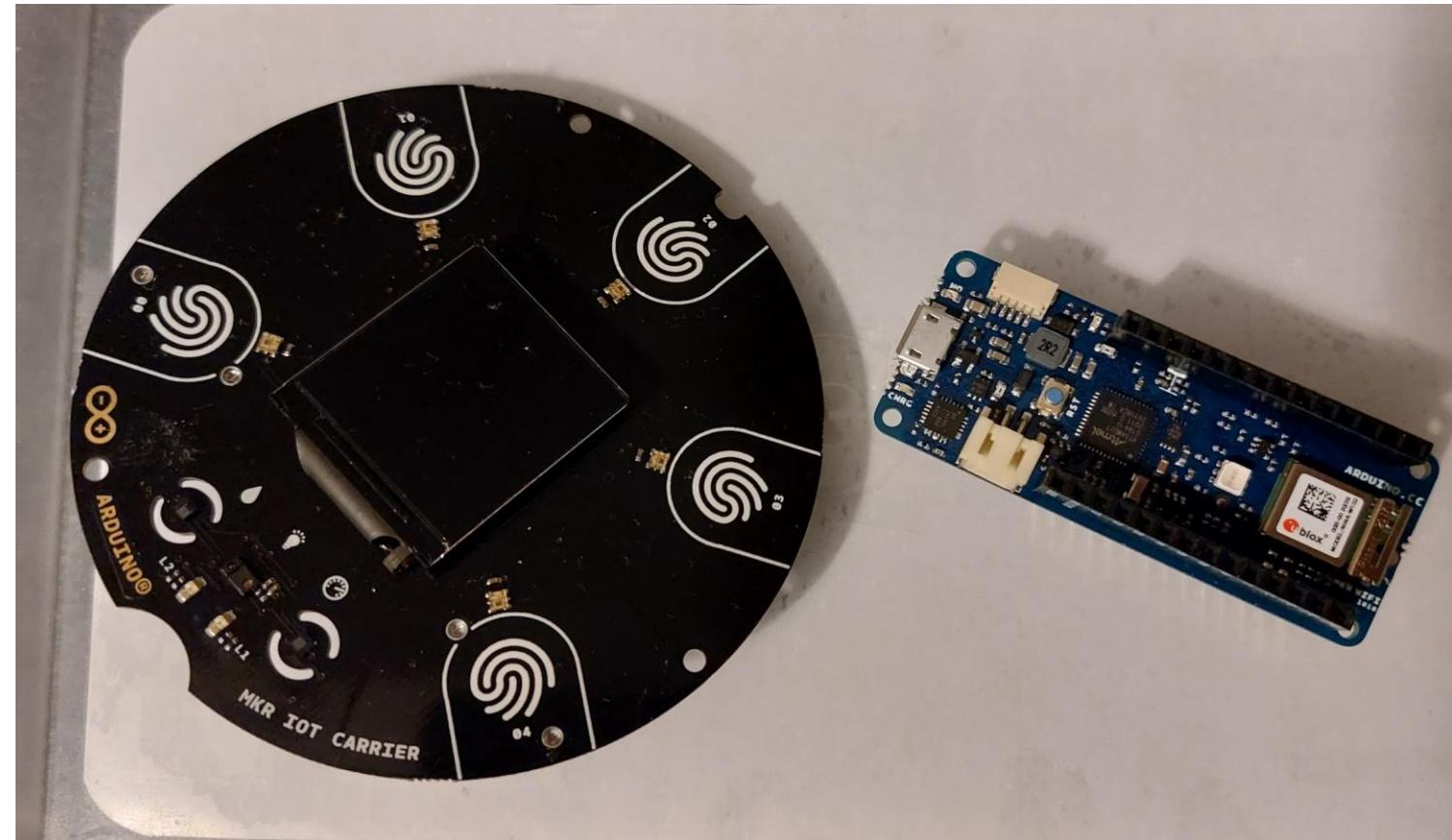
<https://docs.arduino.cc/hardware/mkr-wifi-1010>

Arduino MKR WiFi 1010

- **Microcontrollore:** SAMD21 Cortex-M0+ 32bit low power ARM MCU
- **Alimentazione (USB/VIN):** 5 VDC
- **Supporta alimentazione a batteria:** 1 batteria Li-Po a singola cella da 3,7 Volt – minimo 700 mAh
- **Tensione operativa:** 3,3V
- **I/O digitali:** 8
- **PWM:** 12 (pin 0, 1, 2, 3, 4, 5, 6, 7, 8, 10, A3 o 18, A4 o 19)
- **UART:** 1
- **SPI:** 1
- **I2C:** 1
- **Connettività:** Wi-Fi
- **Ingressi analogici:** 7 (ADC 8/10/12 bit)
- **Uscite analogiche:** 1 (DAC 10 bit)
- **Interrupt esterni:** 8 (0, 1, 4, 5, 6, 7, 8, A1 o 16, A2 o 17)
- **Corrente DC per pin I/O:** 7 mA
- **Memoria Flash:** 256 KB
- **SRAM:** 32 KB
- **EEPROM:** No
- **Velocità di Clock:** 48 MHz
- **Dimensioni (mm):** 61,5x25x20,80
- **Peso:** 32 grammi



MkrIoTcarrier



https://www.arduino.cc/reference/en/libraries/arduino_mkriotcarrier/

<https://docs.arduino.cc/hardware/mkr-iot-carrier>

<http://docs.arduino.cc/tutorials/mkr-iot-carrier/mkr-iot-carrier-01-technical-reference>

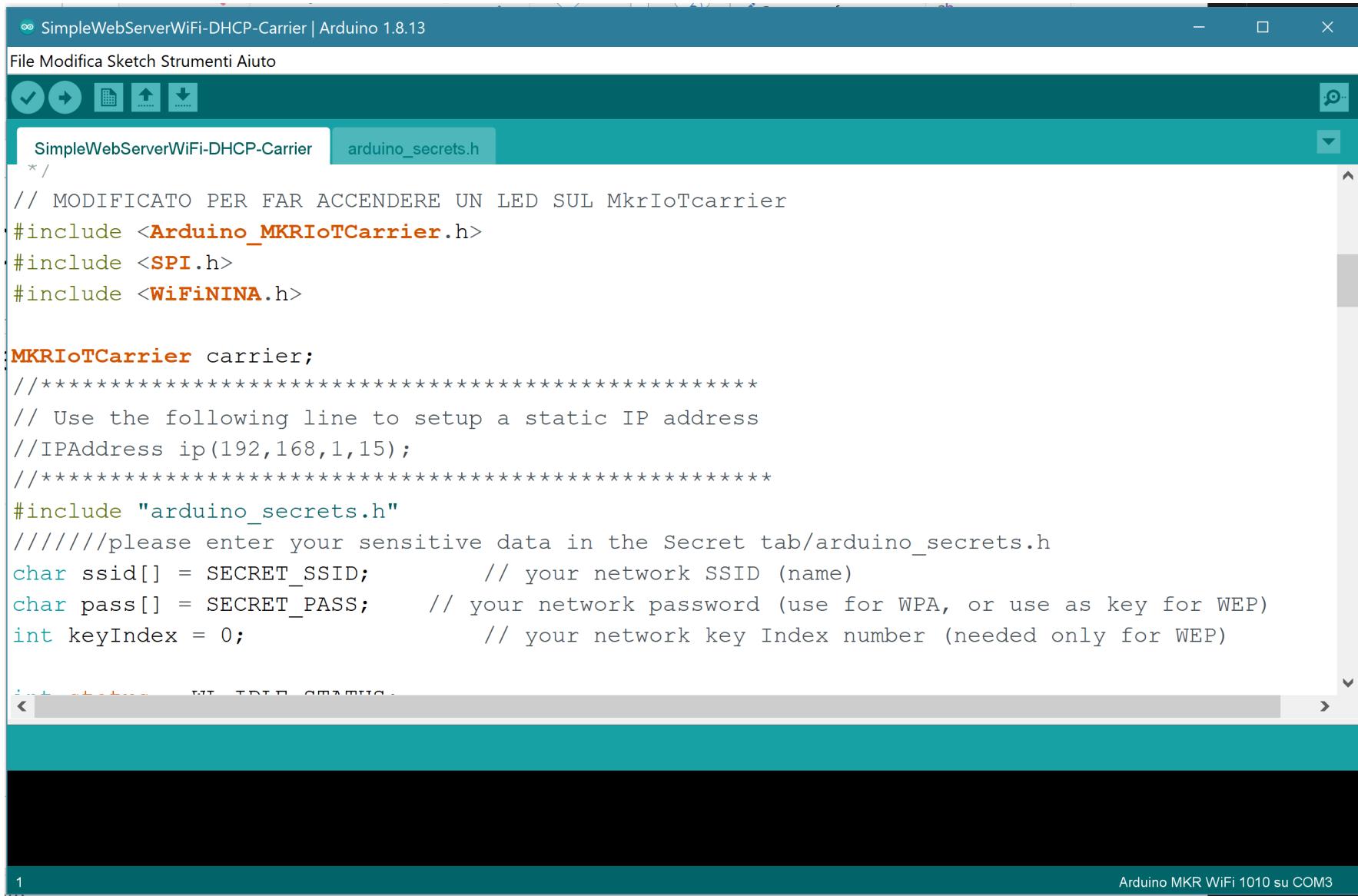
Programmare Arduino

- Installare l'IDE sul proprio PC
- Utilizzare l'IDE on-line

Collegando Arduino alla porta USB è possibile

- Fare l'upload del programma da eseguire
- Comunicare attraverso l'interfaccia seriale con l'IDE

L'IDE di Arduino



The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** Shows the project name "SimpleWebServerWiFi-DHCP-Carrier" and the version "Arduino 1.8.13".
- Menu Bar:** Includes "File", "Modifica", "Sketch", "Strumenti", and "Aiuto".
- Toolbar:** Features icons for file operations (New, Open, Save, Print, Upload, Download) and a search function.
- Sketch Editor:** Displays the code for "SimpleWebServerWiFi-DHCP-Carrier". The tab "arduino_secrets.h" is currently selected.
- Code Content:** The code includes headers for MKR IoT Carrier, SPI, and WiFi NINA, and defines a carrier object and network settings. It also includes a comment about entering sensitive data in the "arduino_secrets.h" file.
- Bottom Status Bar:** Shows the port "Arduino MKR WiFi 1010 su COM3" and the number "1".

Esempio 1: connessione alla WiFi

Utilizza alcune librerie:

```
#include <SPI.h>
```

```
#include <WiFiNINA.h>
```

E per interfacciarsi con lo shield (il Carrier)

```
#include <Arduino_MKRIoTCarrier.h>
```

setup e loop

I programmi comprendono una funzione setup(), eseguita una sola volta all'avvio, e una funzione loop() ripetuta all'infinito.

```
WiFiServer server(80);

void setup() { ... status = WiFi.begin(ssid, pass);
               ... server.begin(); }

void loop() { ... WiFiClient client = server.available();

               while (client.connected()) {           // loop while the client's connected
                   if (client.available()) {         // if there's bytes to read from the client,
                       char c = client.read();
                       ...
                   } // legge una riga alla volta = richiesta proveniente dal client (via browser)
                   client.print("Click <a href=\"/H\">here</a> turn the LED on pin LED_BUILTIN on<br>");
                   client.print("Click <a href=\"/L\">here</a> turn the LED on pin LED_BUILTIN off<br>");
                   if (currentLine.endsWith("GET /H")) {
                       Carrier.leds.setPixelColor(0, color); // set Led 0 to color "color"
                       carrier.leds.show();                  // Refresh strip
                   }
               }
               client.stop();
}
```

Esempio 2: comunicazione con MQTT

```
#include <MQTT.h>

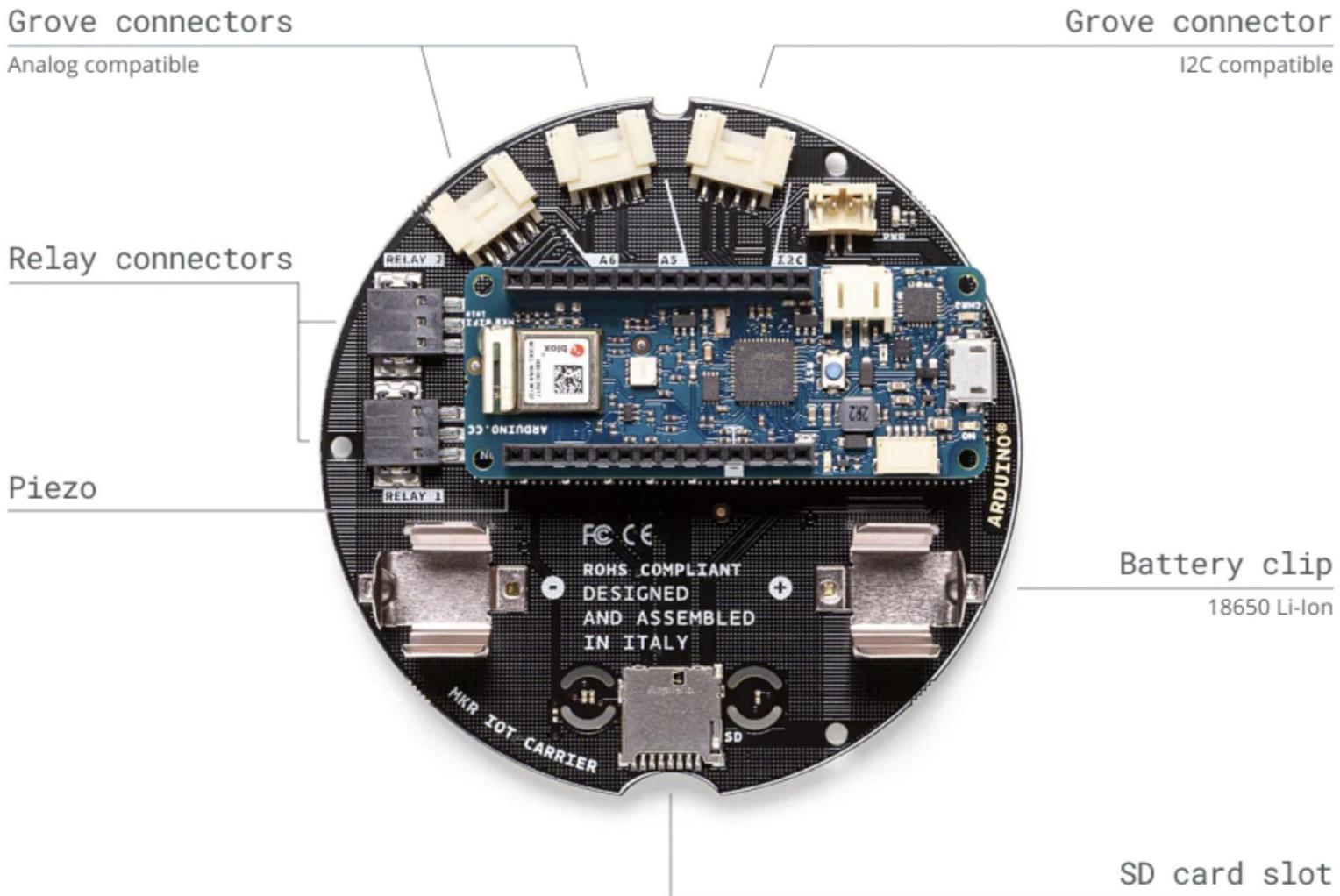
MQTTClient client;

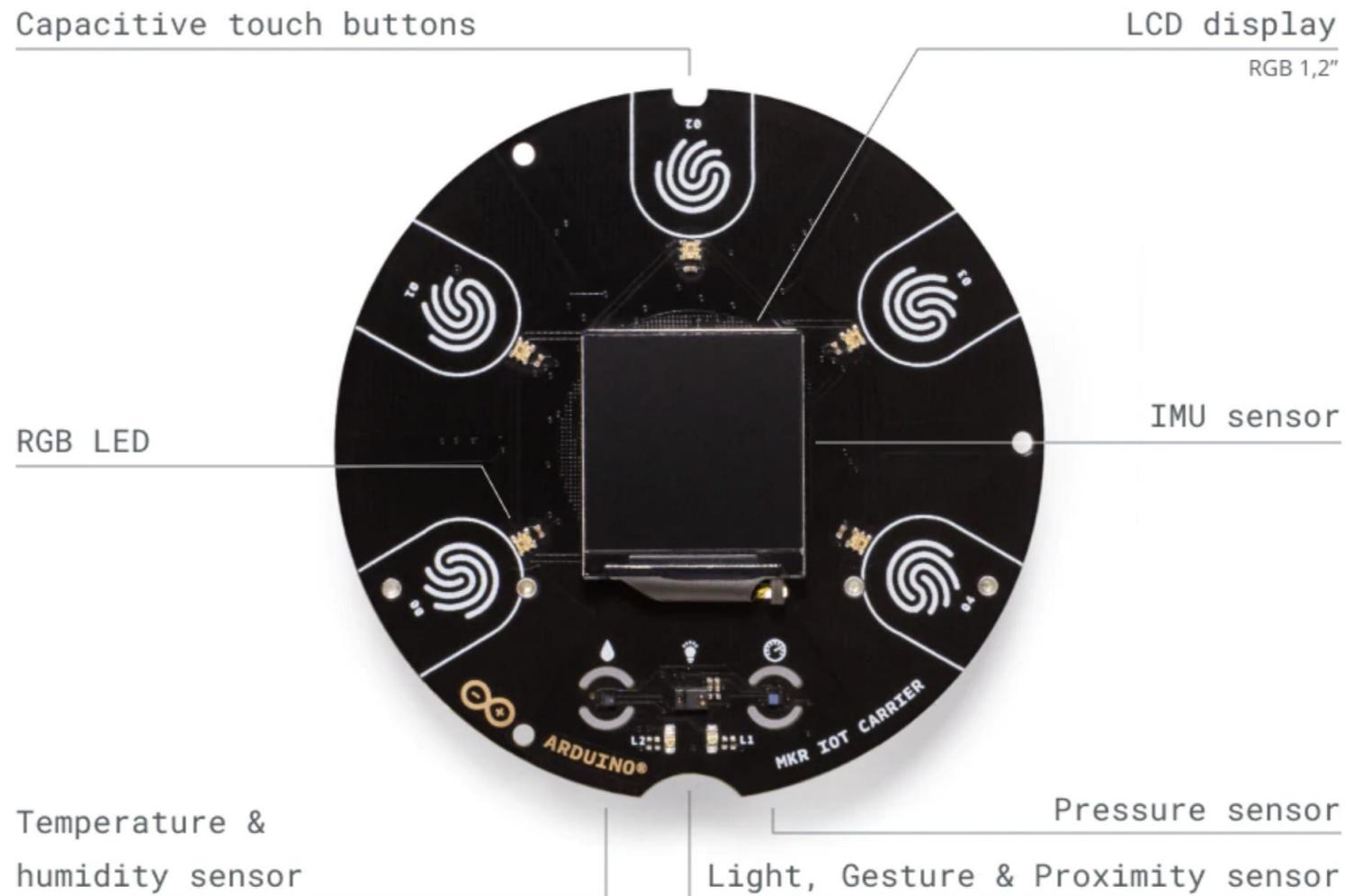
#define BROKER_IP  "193.206.52.98"
#define DEV_NAME   "mqttdevice"
#define MQTT_USER  "pissir"
#define MQTT_PW    "pissir2020"

...
void setup() {  client.begin(BROKER_IP, 1883, net);
                client.onMessage(messageReceived);
                ... client.connect(DEV_NAME, MQTT_USER, MQTT_PW);
                ... client.subscribe("/hello");}
void loop() { ... client.publish("/hello", "world"); ... }
```

Esempio 2: comunicazione con MQTT (cont.)

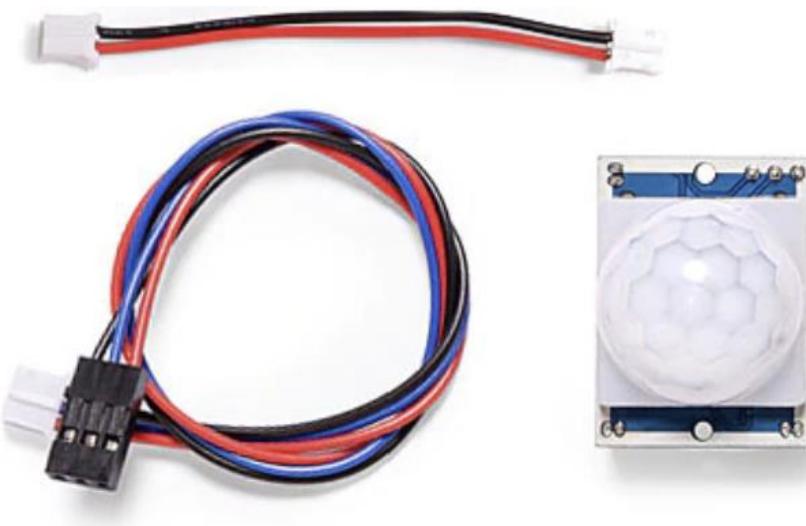
```
void messageReceived(String &topic, String &payload) {  
    Serial.println("incoming: " + topic + " - " + payload);  
    if (topic == "/hello") {  
        if (payload == "open") {  
            Serial.println("open");  
            carrier.leds.setPixelColor(0, color); // set Led 0 to color "color"  
            carrier.leds.show();           // Refresh strip  
        } else if (payload == "closed") {  
            Serial.println("closed");  
            carrier.leds.setPixelColor(0,0); // switch Led 0 off  
            carrier.leds.show();  }  
    }  
}
```

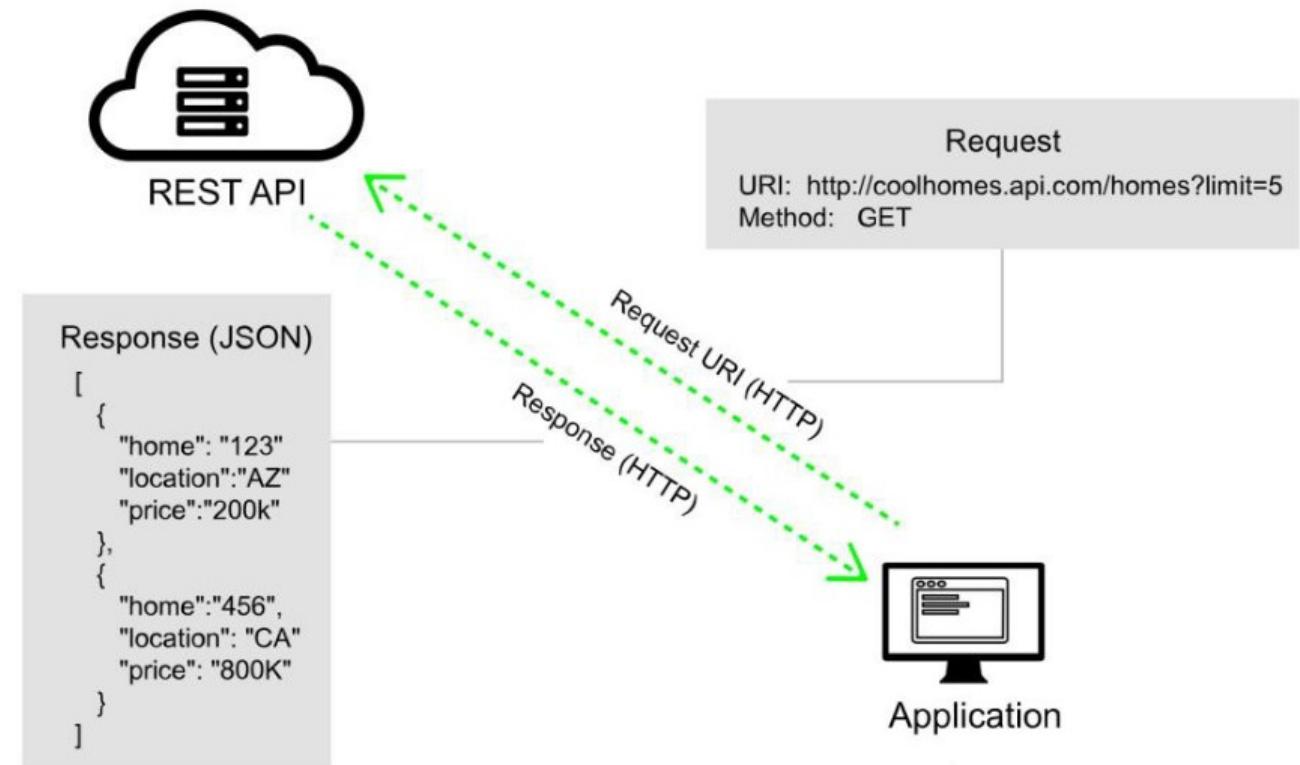




Due moduli grove esterni:

- Per misurare l'umidità del terreno
- Per rilevare il movimento





REST&JSON

The REpresentational State Transfer (REST) style and the JavaScript Object Notation (JSON) data interchange format

- A style of software architecture for distributed systems
- Platform-independent
 - you don't care if the server is Unix, the client is a Mac, or anything else
- Language-independent
 - C# can talk to Java, etc...
- Standards-based
 - runs on top of HTTP
- Can easily be used in the presence of firewalls

- A resource can be anything that has identity
 - a document or image
 - a service, e.g., "today's weather in New York"
 - a collection of other resources
 - non-networked objects (e.g., people)
- The resource is the **conceptual mapping** to an entity or set of entities, not necessarily the entity that corresponds to that mapping at any particular point in time!

- Resource: source of specific information
- Mapping: Resources \Leftrightarrow URIs
- Client and server exchange representations of the resource
 - the same resource may have different representations
 - e.g., XML, JSON, HTML, RDF, ...
- Operations on the Resource is done by means of HTTP methods
 - GET, POST, PUT, DELETE
 - Mapping to CRUD: create, read, update, delete

- **Collection** resource
 - Represents a set (or list) of items
 - Format: /resource
 - e.g., <http://api.uniupo.it/students>
<http://api.uniupo.it/courses>
- **Element(Item)** resource
 - Represents a single item, and its properties
 - Format: /resource/identifier
 - e.g., <http://api.uniupo.it/students/s123456>
<http://api.uniupo.it/courses/S1729>

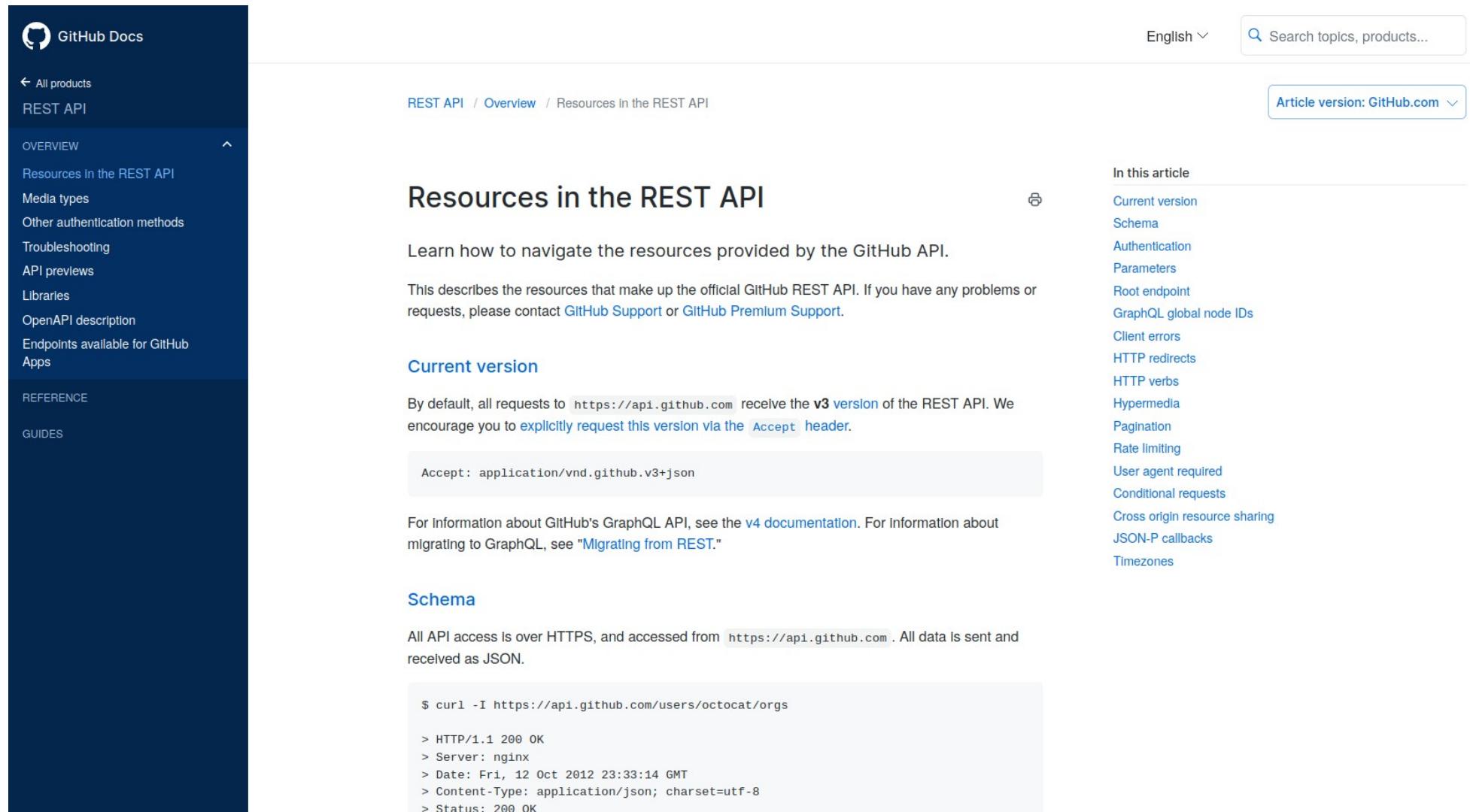
- Nouns (not verbs)
- Plural nouns
- Concrete names (not abstract)
 - /courses, not /items

- **GET**
 - Retrieve the representation of the resource (in the HTTP response body)
 - Collection: the list of items
 - Element: the properties of the element
- **POST**
 - Create a new resource (data in the HTTP request body)
 - Use a URI for a Collection
- **PUT**
 - Update an existing element (data in the HTTP request body)
 - Mainly for elements' properties
- **DELETE**

RESOURCE	GET	POST	PUT	DELETE
/dogs	List dogs	Create a new dog	Bulk update dogs(avoid)	Delete all dogs(avoid)
/dogs/1234	Show info about thedog with id 1234	ERROR	If exists, update the info about dog #1234	Delete the dog #1234

- A given Element may have a (1:1 or 1:N) relationship with other Element(s)
- Represent with: /resource/identifier/resource
- e.g.,
<http://api.uniupo.it/students/s123456/courses>
<http://api.uniupo.it/courses/S1729/students>

- Returned in GET, sent in PUT/POST
- Different formats are possible
- Mainly: XML, JSON
 - But also: SVG, JPEG, TXT, ...
 - In POST: URL-encoding
- Format may be specified in
 - Request headers
 - Accept: application/json
 - URI extension
 - `http://api.uniupo.it/students/s123456.json`
 - Request parameter
 - `http://api.uniupo.it/students/s123456?format=json`



The screenshot shows the GitHub REST API documentation page. The left sidebar has a dark blue background with white text, listing categories like All products, REST API, OVERVIEW, and REFERENCE. The main content area has a white background with a red header bar at the top. The header bar includes the GitHub Docs logo, a search bar with placeholder "Search topics, products...", and language selection for English. The main title is "Resources in the REST API". Below it, there's a section titled "Current version" with a note about the v3 version and how to request it via the Accept header. Another section titled "Schema" shows a curl command example. On the right side, there's a sidebar titled "In this article" with a list of topics including Current version, Schema, Authentication, Parameters, Root endpoint, GraphQL global node IDs, Client errors, HTTP redirects, HTTP verbs, Hypermedia, Pagination, Rate limiting, User agent required, Conditional requests, Cross origin resource sharing, JSON-P callbacks, and Timezones.

Resources in the REST API

Learn how to navigate the resources provided by the GitHub API.

This describes the resources that make up the official GitHub REST API. If you have any problems or requests, please contact [GitHub Support](#) or [GitHub Premium Support](#).

Current version

By default, all requests to `https://api.github.com` receive the [v3 version](#) of the REST API. We encourage you to [explicitly request this version via the `Accept` header](#).

```
Accept: application/vnd.github.v3+json
```

For information about GitHub's GraphQL API, see the [v4 documentation](#). For information about migrating to GraphQL, see "[Migrating from REST](#)."

Schema

All API access is over HTTPS, and accessed from `https://api.github.com`. All data is sent and received as JSON.

```
$ curl -I https://api.github.com/users/octocat/orgs
> HTTP/1.1 200 OK
> Server: nginx
> Date: Fri, 12 Oct 2012 23:33:14 GMT
> Content-Type: application/json; charset=utf-8
> Status: 200 OK
```

<https://docs.github.com/en/rest/overview/resources-in-the-rest-api>

Documentation

 Search the docs

Getting started

Fundamentals



Tools and libraries

Tutorials

API reference Index

Twitter API

Twitter Ads API

Twitter for Websites

Twitter Developer Labs

API reference index

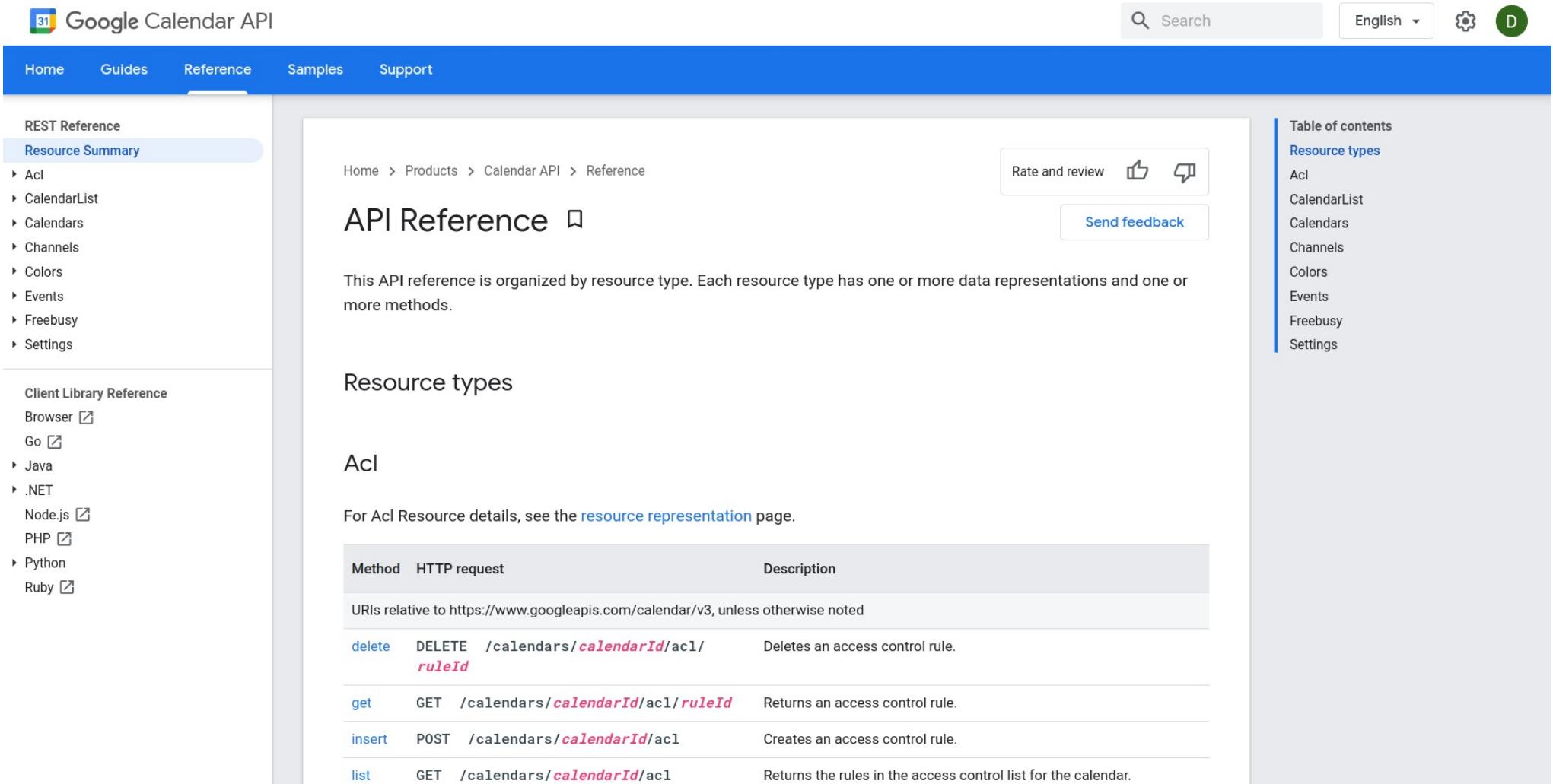
The API reference index is a central list of all endpoints included on the Twitter Developer Platform across our different APIs.

Jump to...

- [Twitter API v2](#)
- [Twitter API - Enterprise](#)
- [Twitter API - Premium v1.1](#)
- [Twitter API - Standard v1.1](#)
- [Twitter Ads API](#)
- [Labs](#)
- [Platform-wide](#)

Twitter API v2: Early Access

<https://developer.twitter.com/en/docs/api-reference-index>



The screenshot shows the Google Calendar API Reference page. At the top, there's a navigation bar with links for Home, Guides, Reference, Samples, and Support. The 'Reference' tab is selected. On the left, a sidebar lists 'REST Reference' with 'Resource Summary' selected, and 'Client Library Reference' with links for various languages like Browser, Go, Java, .NET, Node.js, PHP, Python, and Ruby. The main content area has a breadcrumb trail: Home > Products > Calendar API > Reference. It features a 'Rate and review' button with thumbs up and down icons, and a 'Send feedback' button. The title 'API Reference' is followed by a brief description: 'This API reference is organized by resource type. Each resource type has one or more data representations and one or more methods.' Below this, a section titled 'Resource types' lists 'Acl'. For the 'Acl' resource, it says: 'For Acl Resource details, see the [resource representation](#) page.' A table then lists the methods for interacting with the Acl resource:

Method	HTTP request	Description
delete	DELETE /calendars/ <i>calendarId</i> /acl/ <i>ruleId</i>	Deletes an access control rule.
get	GET /calendars/ <i>calendarId</i> /acl/ <i>ruleId</i>	Returns an access control rule.
insert	POST /calendars/ <i>calendarId</i> /acl	Creates an access control rule.
list	GET /calendars/ <i>calendarId</i> /acl	Returns the rules in the access control list for the calendar.

<https://developers.google.com/calendar/v3/reference>

The screenshot shows the Philips Hue developer website. At the top, there's a navigation bar with links for Explore, Develop, Support, Job Vacancies, Forum, and Login. The main heading is "How to develop for Hue?". Below the heading, there are several navigation links: Develop, Get Started, Application Design Guidance, Hue API, Hue Entertainment, and Tools and SDKs. The background of the page features a photograph of two people sitting on the floor, looking at a screen and writing in a notebook.

Get Started

Core Concepts

The fastest way to learn how to build apps which control the hue system is to use the simple test web app built into every bridge. This lets you directly input commands and send them to the lights. You can look at the source HTML and JavaScript code for directions on how to do something different.

Follow 3 Easy Steps

Step 1

First make sure your bridge is connected to your network and is functioning properly. Test that the smartphone app can control the lights on the same network.

Step 2

Then you need to discover the IP address of the bridge on your network. You can do this in a few ways.

On this page:

[Follow 3 Easy Steps](#)

[So let's get started...](#)

[Turning a light on and off](#)

<https://developers.meethue.com/develop/get-started-2/>
Login required

REST & JSON

- Use ?parameter=value for more advanced resource filtering (or search)
 - E.g.,
 - [https://api.twitter.com/1.1/statuses/user_timeline.json
?screen_name=twitterapi&count=2](https://api.twitter.com/1.1/statuses/user_timeline.json?screen_name=twitterapi&count=2)

- When errors or exceptions are encountered, use meaningful HTTP Status Codes
 - The Response Body may contain additional information (e.g., informational error messages)

```
{  
  "developerMessage" : "Verbose, plain language description of  
  the problem for the developer with hints about how to fix it.",  
  "userMessage" : "Pass this message on to the app user if  
  needed.",  
  "errorCode" : 12345,  
  "more info" : "http://dev.teachdogrest.com/errors/12345"  
}
```

- Twitter Streaming API

Authorization: Oauth version 1 and 2

oauth_consumer_key="xvz1evFS4wEEPTGEFPHBog", ...

- Amazon Web Services API

Authorization: AWS based on keyed-HMAC (Hash Message Authentication Code)

AKIAIOSFODNN7EXAMPLE:frJIUNo//yllqDzg=

- Google API

Authorization: Oauth version 2 from the Google API Console

URL Design

Plural nouns for collections	/dogs
ID for entity	/dogs/1234
Associations/	owners/5678/dogs
HTTP Methods	POST GET PUT DELETE
Bias toward concrete names	/dogs (not animals)
Multiple formats in URL	/dogs.json /dogs.xml
Paginate with limit and offset	?limit=10&offset=0
Query params	?color=red&state=running
Partial selection	?fields=name,state
Use medial capitalization	"createdAt": 1320296464myObject.createdAt;
Use verbs for non-resource requests	/convert?from=EUR&to=CNY&amount=100
Search	/search?q=happy%2Blabrador

Versioning

Include version in URL	/v1/dogs
Keep one previous version long enough for developers to migrate	/v1/dogs /v2/dogs

Errors

Status Codes	200 201 304 400 401 403 404 500
Verbose messages	{"msg":"verbose, plain language hints"}

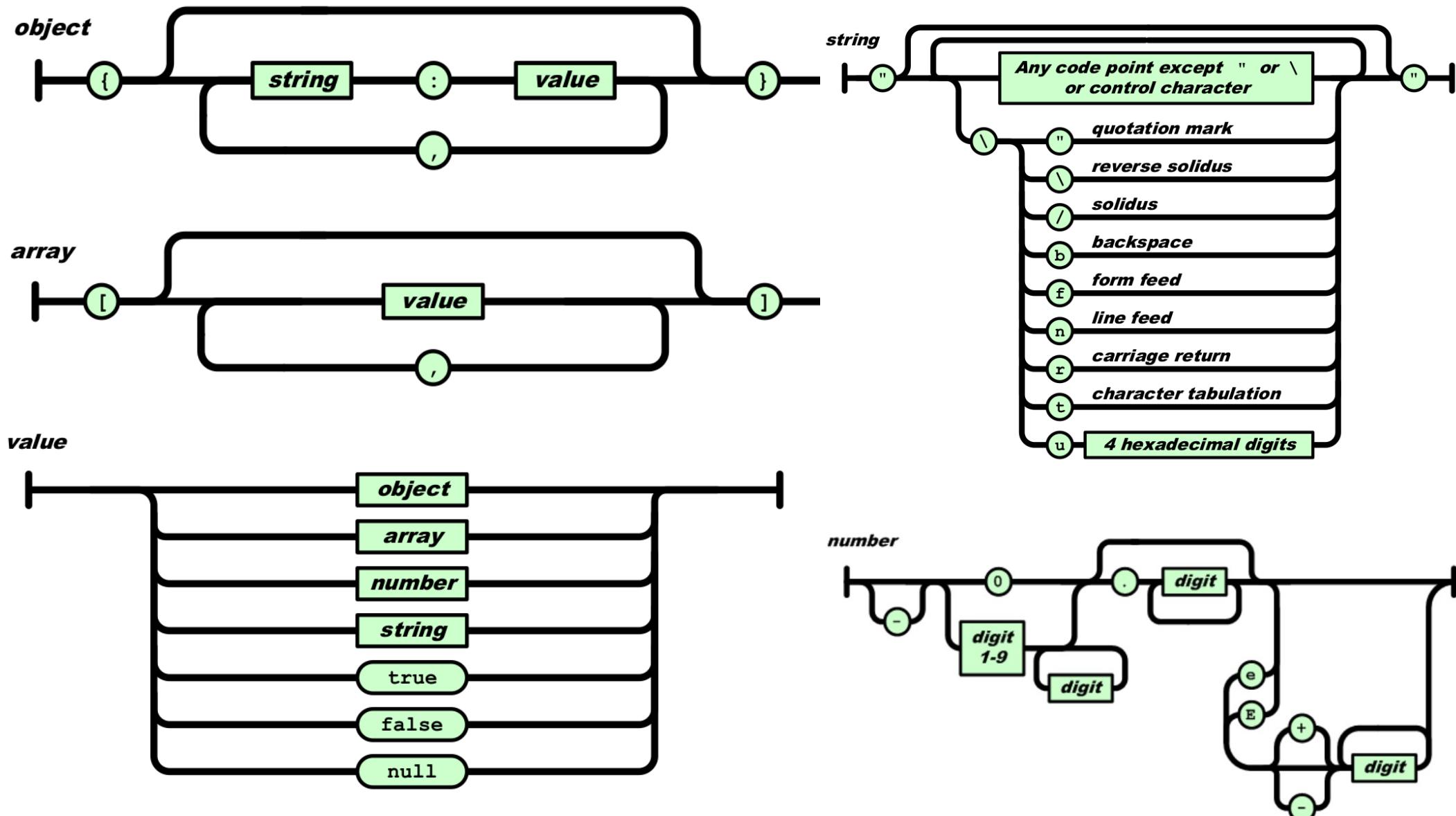
- “JSON (JavaScript Object Notation) is a lightweight data interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate”

-JSON.org

- Important:
 - JSON is a subset of JavaScript

- JSON is built on two structures:
 - A **collection** of name/value pairs. In various languages, this is realized as an **object**, record, struct, dictionary, hash table, keyed list, or associative array.
 { ... }
 - An **ordered list** of values. In most languages, this is realized as an **array**, vector, list, or sequence.
 [...]

```
{  
    "firstName": "John", } } Name/Value Pairs  
    "lastName": "Smith", } } Child properties  
    "address": {  
        "streetAddress": "21 2nd Street",  
        "city": "New York",  
        "state": "NY",  
        "postalCode": 10021  
    },  
    "phoneNumbers": [  
        "212 555-1234", } String Array  
        "646 555-4567" } Number data type  
    ]  
}
```



- REST

- http://en.wikipedia.org/wiki/Representational_state_transfer
- R. Fielding, Architectural Styles and the Design of Network-based Software Architectures,
<https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
- <https://pages.apigee.com/ebook-web-api-design-registration.html>
- <http://www.slideshare.net/apigee/api-design-3rd-edition>
- <https://cloud.google.com/apis/design/>

- JSON
 - <http://json.org>
 - ECMA-404 The JSON Data Interchange Standard
<https://www.ecma-international.org/publications-and-standards/standards/ecma-404/>

- This work is licensed under the Creative Commons “Attribution-NonCommercial-ShareAlike Unported(CC BY-NC-SA 4.0)” License.
- You are free:
 - to Share-to copy, distribute and transmit the work
 - to Remix-to adapt the work
- Under the following conditions:
 - Attribution-You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
 - Noncommercial-You may not use this work for commercial purposes.
 - Share Alike-If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.
- To view a copy of this license, visit
 - <https://creativecommons.org/licenses/by-nc-sa/4.0/>
- This work is a derivative of works by Luigi De Russis and Elia Callegari



Building Web Applications

WITH SPARK

Introduction to the Spark framework. A practical example: a REST server for tasks management

Goal

- Create REST services
 - in Java
- Learn a simple framework
 - but extensible

Spark

- "*A micro framework for creating web applications in Java 8 with minimal effort*"
- <http://sparkjava.com/>
- Mainly used for creating REST APIs
- Support
 - session, cookie, redirect, ...
- Extensible
 - no HTML templating engine, ORM, etc. included
 - various packages to choose from

Spark Installation

- Use gradle

```
implementation "com.sparkjava:spark-  
core:2.8.0"
```

- or Maven

```
<dependency>  
    <groupId>com.sparkjava</groupId>  
    <artifactId>spark-core</artifactId>  
    <version>2.8.0</version>  
</dependency>
```

Spark Quick Start

```
import static spark.Spark.*;  
  
public class HelloWorld {  
    public static void main(String[] args) {  
        get("/hello", (req, res) -> "Hello World");  
    }  
}
```

- Running the application starts the web server
 - it will run until you kill it

The Web Server

- By default, Spark runs a web server on:
 - <http://127.0.0.1:4567>
- Port can be customized, before anything else:

```
port(8080); // Spark will run on port 8080
```
- You can replace the embedded web server (Jetty) with other servers
 - by adding a configuration in your web.xml
 - <http://sparkjava.com/documentation#embedded-web-server>

Web Resources

- Each resource is implemented by a method

```
get("/", (request, response) -> {  
    return ("Hello World");  
}) ;
```

- Must specify
 - the (local) URL at which the resource will be visible
 - the HTTP verb
 - the content of the resource: return statement

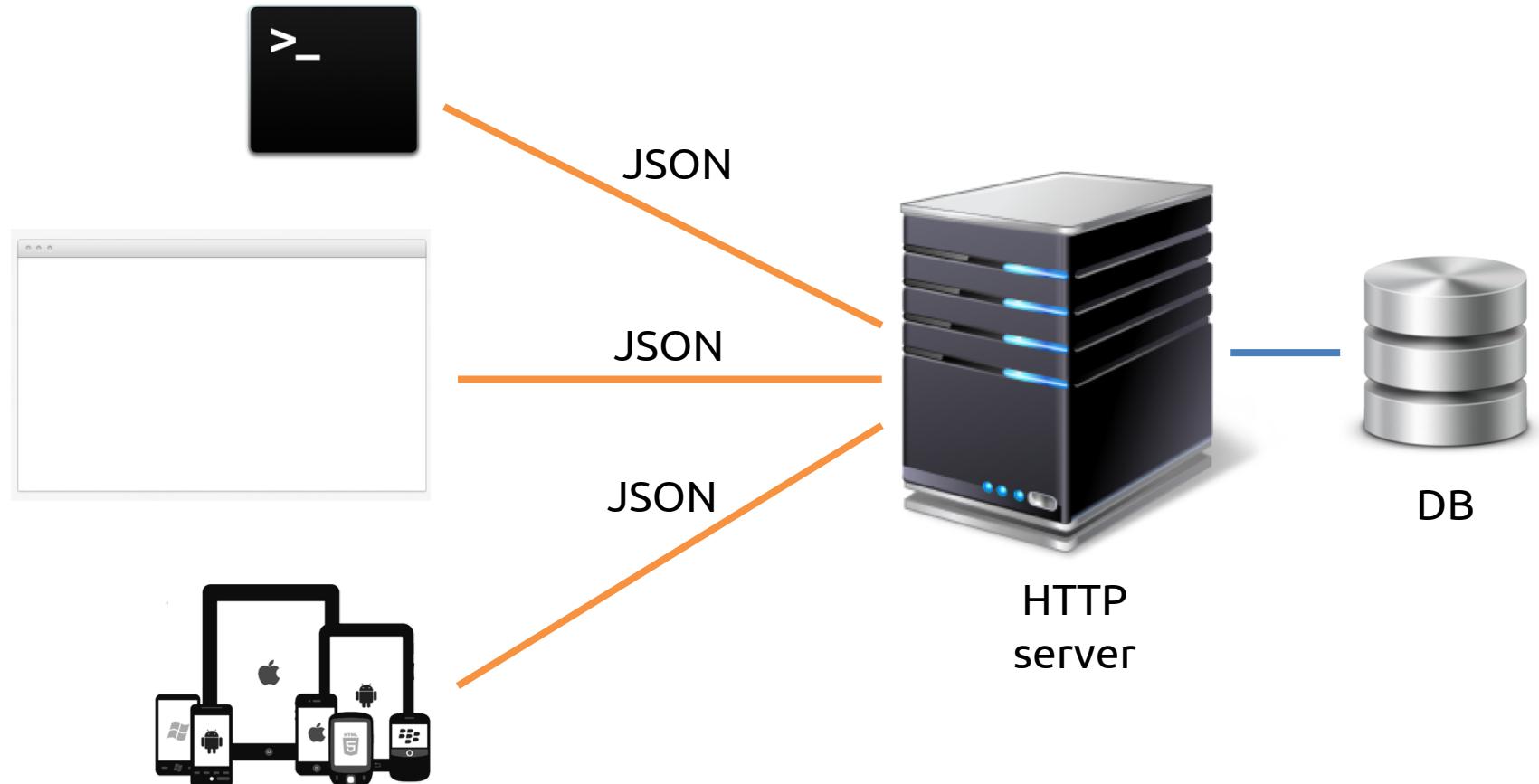
A case study

REST SERVER WITH SPARK

Spark Tasks Server

- A basic server that:
 - Connect to a database that contains a series of tasks
 - For each task, it provides:
 - task description
 - whether the task is "urgent" or not
 - Can show
 - a single task
 - all the existing tasks
 - Can create a new task

Spark Tasks Server



Resources

- Identify the resources to expose
 - Task
 - Represents a single task with its own information
 - Each task is identified by a unique id
 - Tasks
 - The set of tasks available

REST service design

- URLs built on resources
 - /api/v1.0/tasks → the collection of tasks
- How to represent a single task
 - They are identified by a unique id
 - /api/v1.0/tasks/22 → the task with id=22

Tasks

- GET /api/v1.0/tasks
 - return all the tasks in JSON
- GET /api/v1.0/tasks/22
 - return task #22 in JSON
- Example

```
{  
  "task": {  
    "description": "buy a new mouse",  
    "id": 22,  
    "urgent": 1  
  }  
}
```

Tasks

- POST /api/v1.0/tasks
 - create a new task
 - an id is assigned automatically to the newly created tasks
 - task description and priority in JSON
- Example of the HTTP request body

```
{  
  "description" : "This is a new task",  
  "urgent" : 0  
}
```

Result

- Checkout on GitHub
 - <https://github.com/reti2vc-2019/spark-rest>

References

- Spark
 - <http://sparkjava.com/>
- Spark Docs
 - <http://sparkjava.com/documentation>
- Spark Tutorials
 - <http://sparkjava.com/tutorials>

Questions?

MF0223 RETI 2

Luigi De Russis

luigi.derussis@uniupo.it



License

- This work is licensed under the Creative Commons “Attribution-NonCommercial-ShareAlike Unported (CC BY-NC-SA 4.0)” License.
- You are free:
 - to **Share** - to copy, distribute and transmit the work
 - to **Remix** - to adapt the work
- Under the following conditions:
 - **Attribution** - You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
 - **Noncommercial** - You may not use this work for commercial purposes.
 - **Share Alike** - If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.
- To view a copy of this license, visit
<https://creativecommons.org/licenses/by-nc-sa/4.0/>

Git

Introduzione ai VCS

• **A cosa serve un Version Control System?**

- Gestisce i cambiamenti di uno o più file nel corso del tempo
- Ogni file avrà la sua ‘storia’, cioè l’insieme di tutte le sue versioni scritte
- Ogni modifica viene tracciata e mantenuta all’interno di un archivio delle versioni
- **E’ uno strumento fondamentale per i team di sviluppo!**



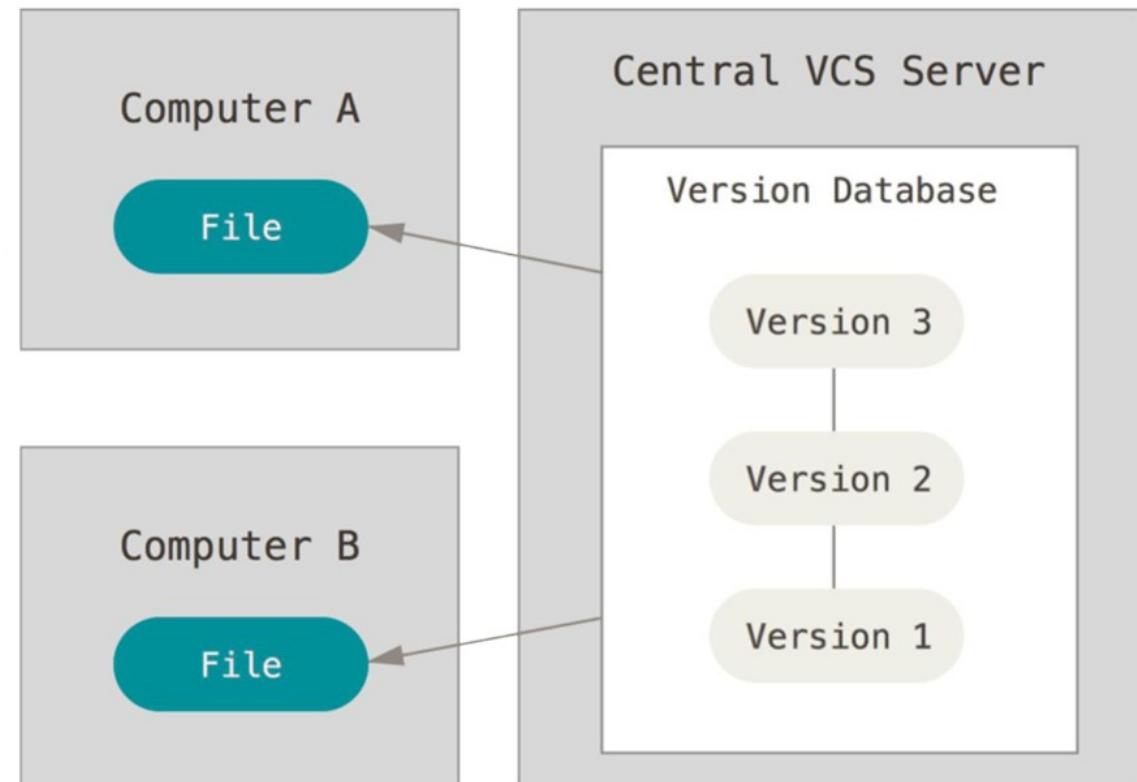
Generale

- Remote Repository: contenitore della versione stabile dei dati
- Workspace/Working directory/Local copy: copia di lavoro dello sviluppatore
- Branch: versione del repository. Non confondere con le versioni dei file!
- Head: stato più recente di un branch (ultimo commit)

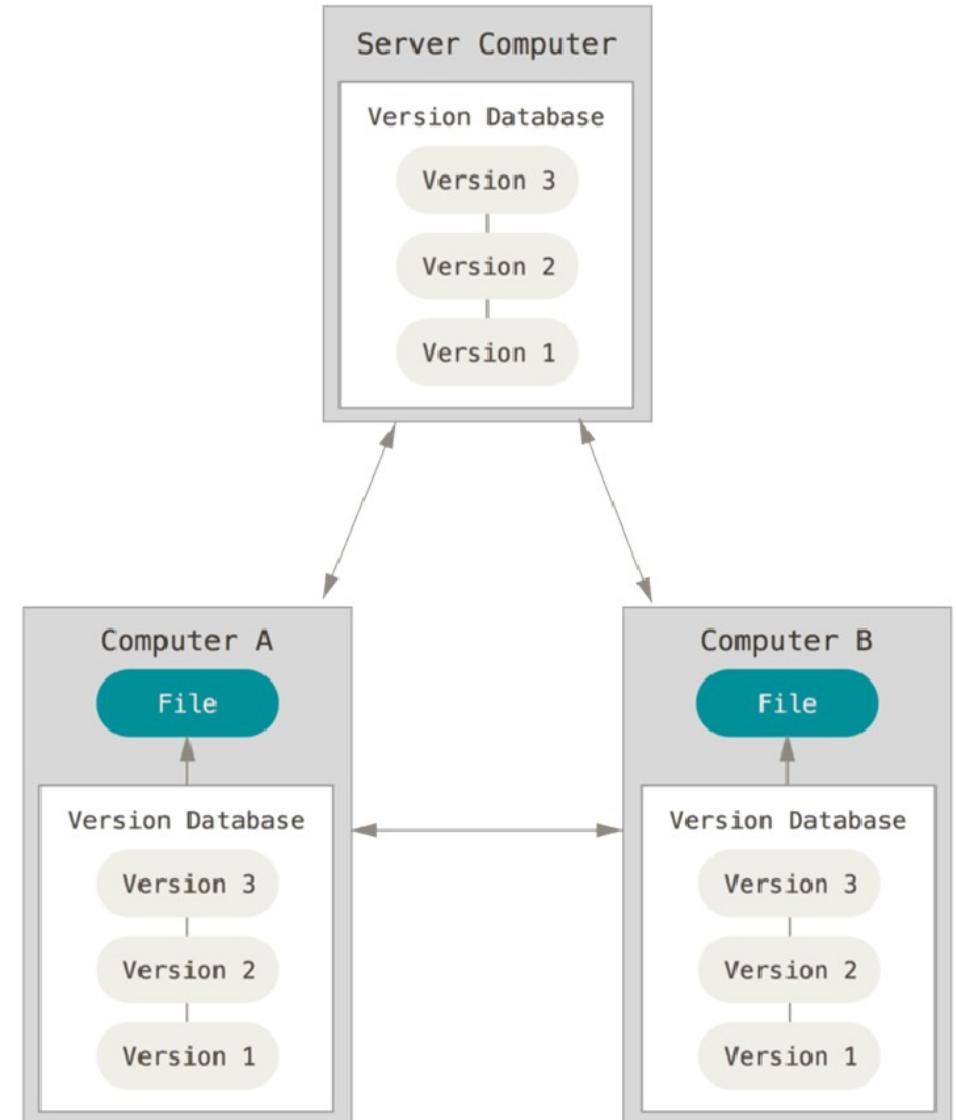
Comandi

- Clone: creazione di una copia del repository
- Commit: inserimento delle modifiche nel local repository
- Push: copia delle modifiche dal repository locale a quello remoto
- Checkout: copia di un branch da repository locale a workspace

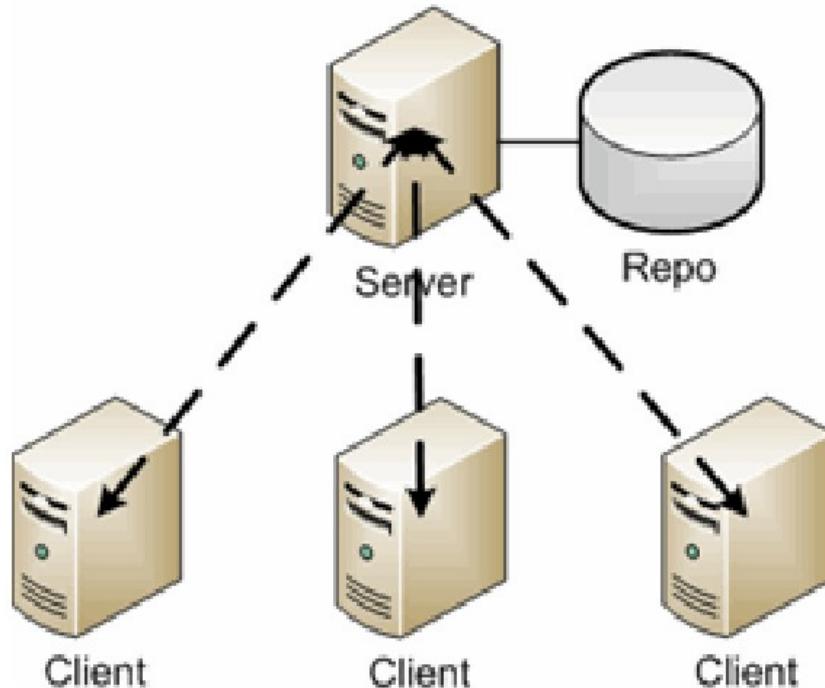
- Un database delle versioni all'interno di un server dedicato
 - Più persone collaborano allo sviluppo di un'applicazione
 - Tracciare le modifiche di tutti gli sviluppatori più semplice
- PROBLEMA: tutti i dati risiedono in un'unica ubicazione.
- Dati inaccessibili per:
 - Malfunzionamenti
 - Mancanza di corrente
 - Errori umani sui repository



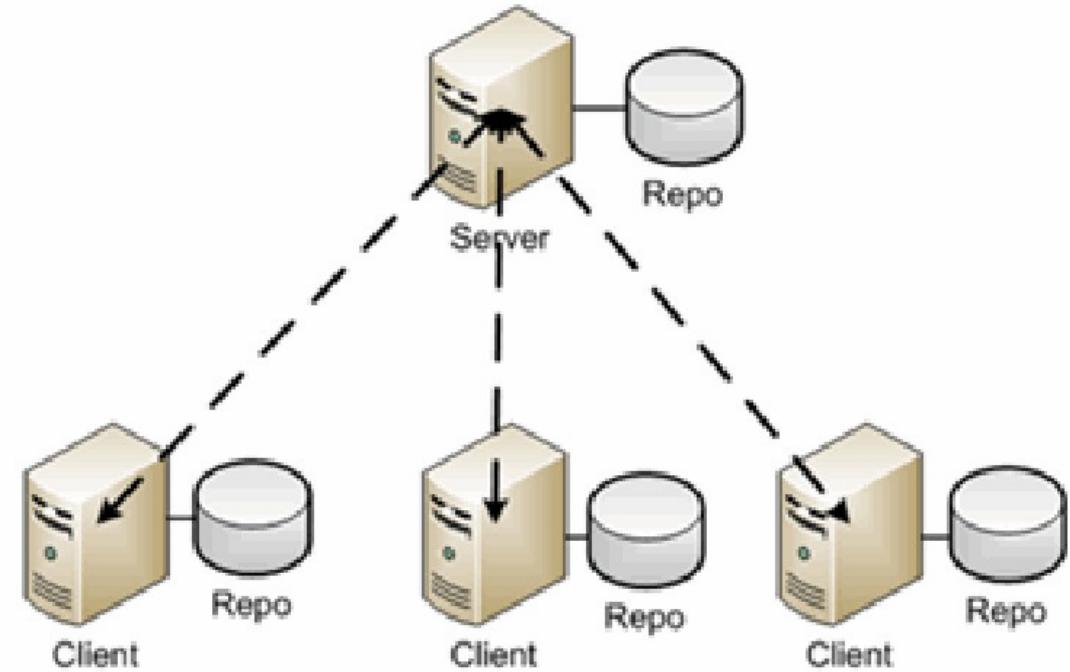
- Il controllo di versione dei repository viene condiviso. In genere tra un server e più client
- Nel server viene mantenuta la copia stabile del codice
- I client effettuano una copia (clone) del repository remoto, creando di fatto un repository locale
- Gli sviluppatori quindi lavoreranno su una working copy del repository locale
- Ogni membro del team interagirà con il repository remoto in modo da non introdurre malfunzionamenti regressioni
- Le copie locali possono essere usate anche per ripristinare il repository remoto nel caso dei file al suo interno fossero corrotti
- La struttura gerarchica client-server viene rimpiazzata da una struttura a ‘nodi’ equiparabili



CVCS

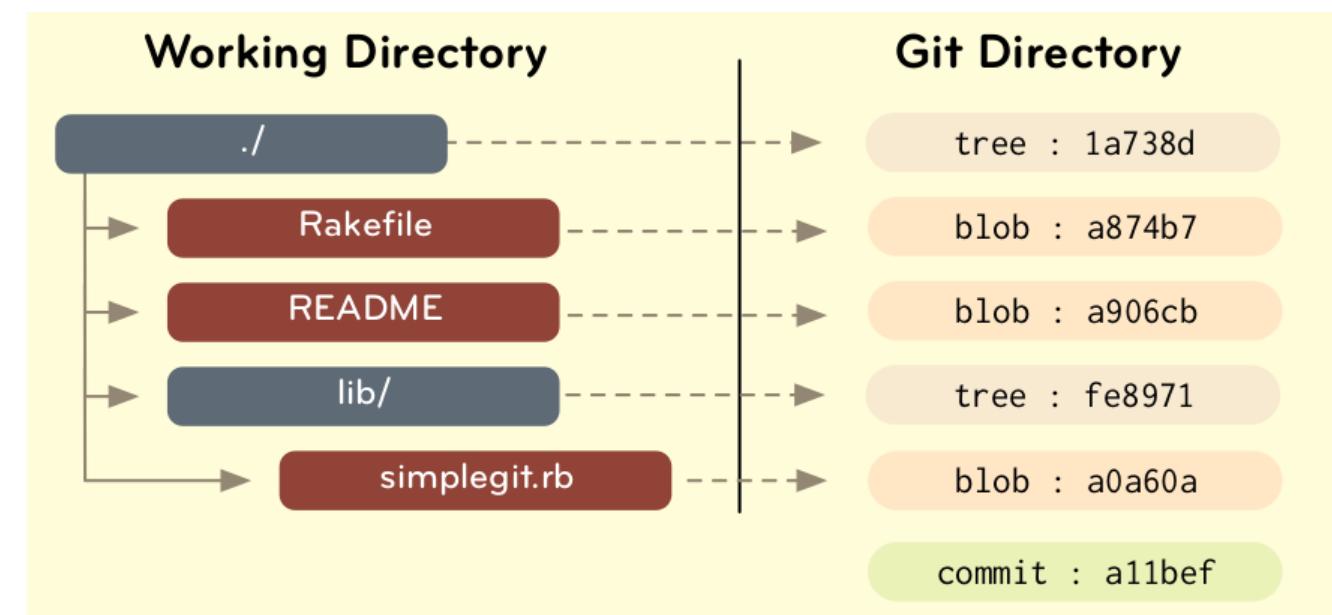
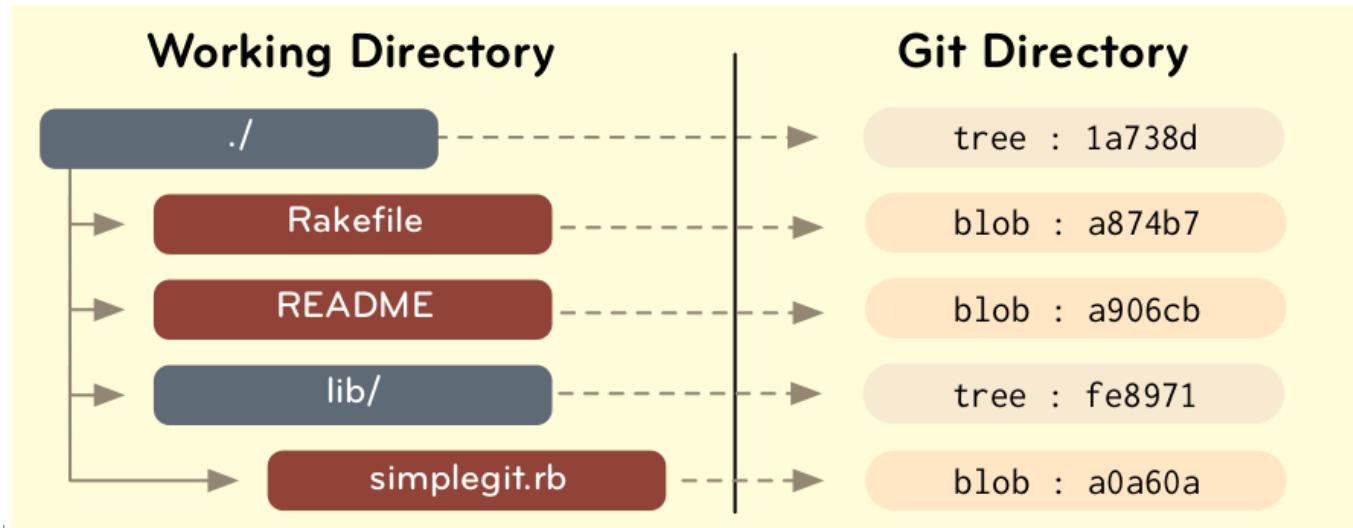


DVCS

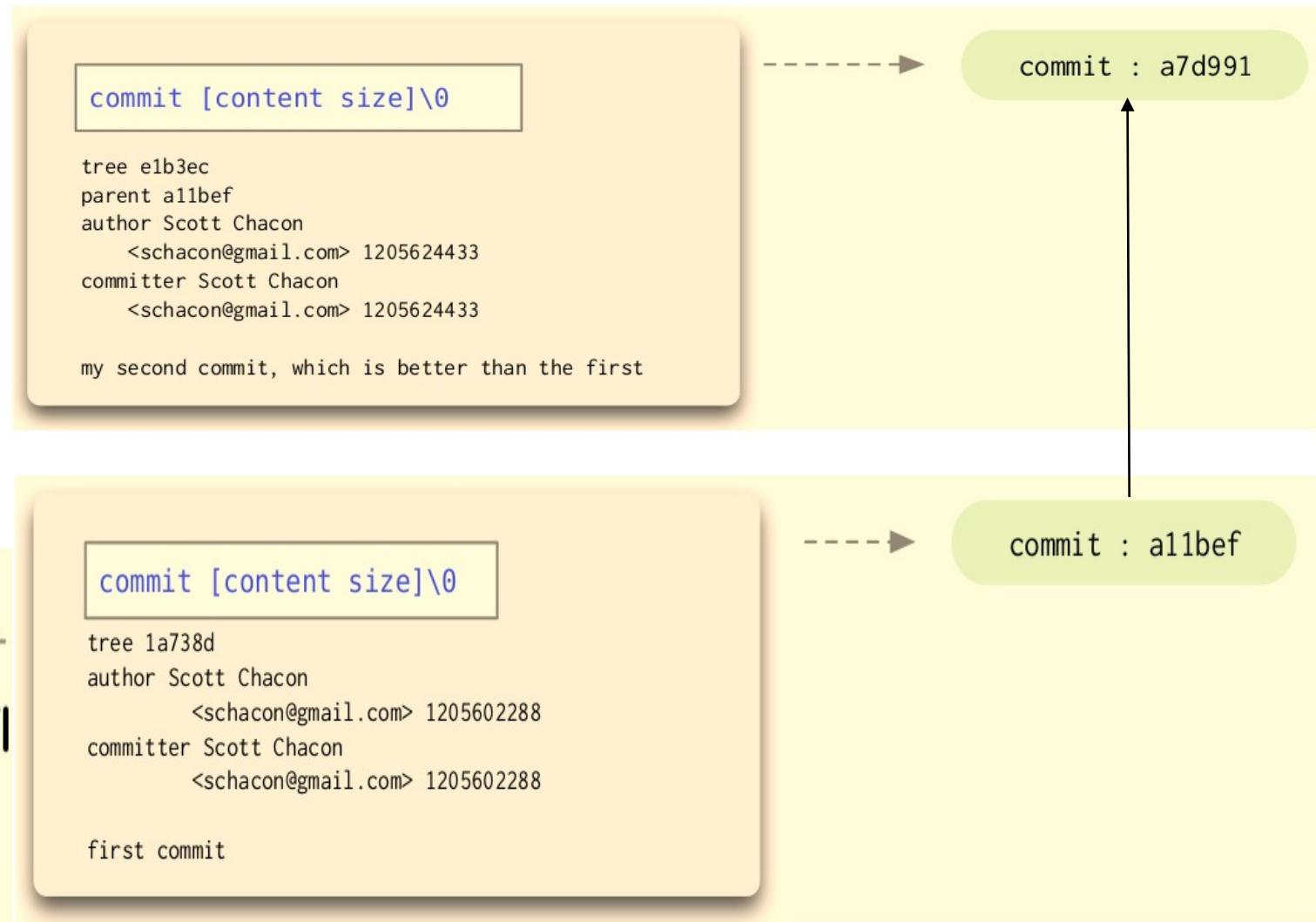
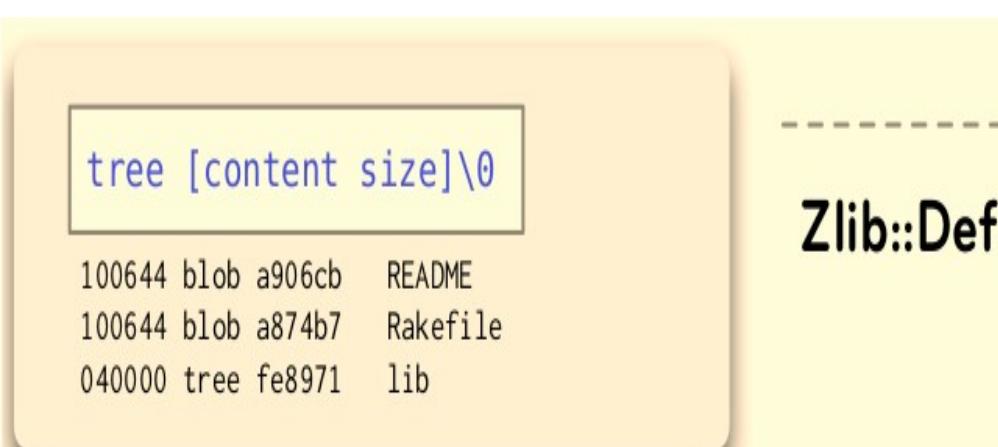


Git - oggetti interni

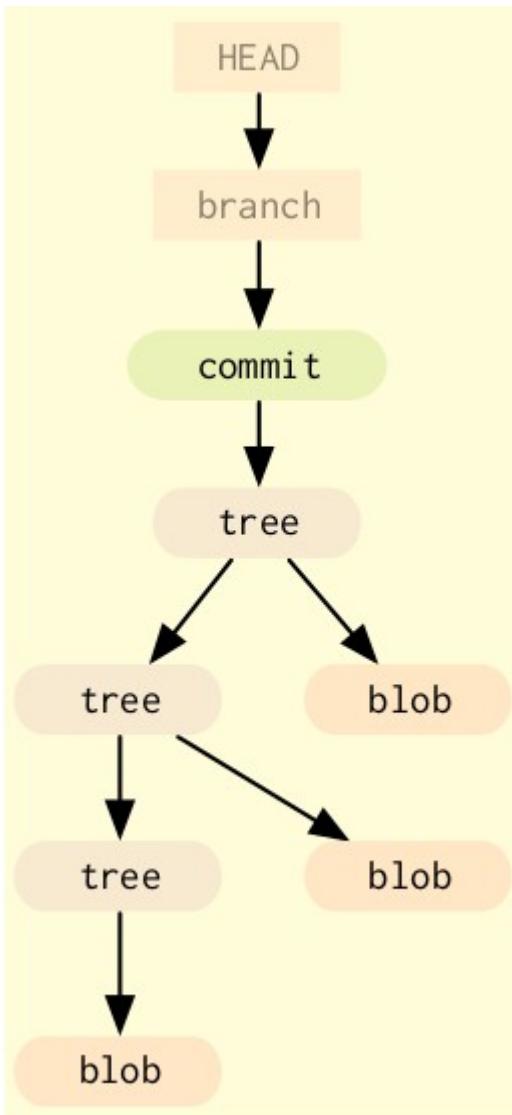
Blob, tree, commit, tag



- Sono contenuti nella directory `.git`
- Sono tutti compressi e identificati dal valore SHA1 del loro contenuto
- Blob
 - Memorizza il contenuto di un file
- Tree
 - Memorizza le directory
- Commit
 - Memorizza una versione di un tree
- Tag
 - Etichetta permanente di uno specifico commit
- Tutti questi oggetti sono **immutabili**
 - Una volta inseriti in Git non cambiano più!

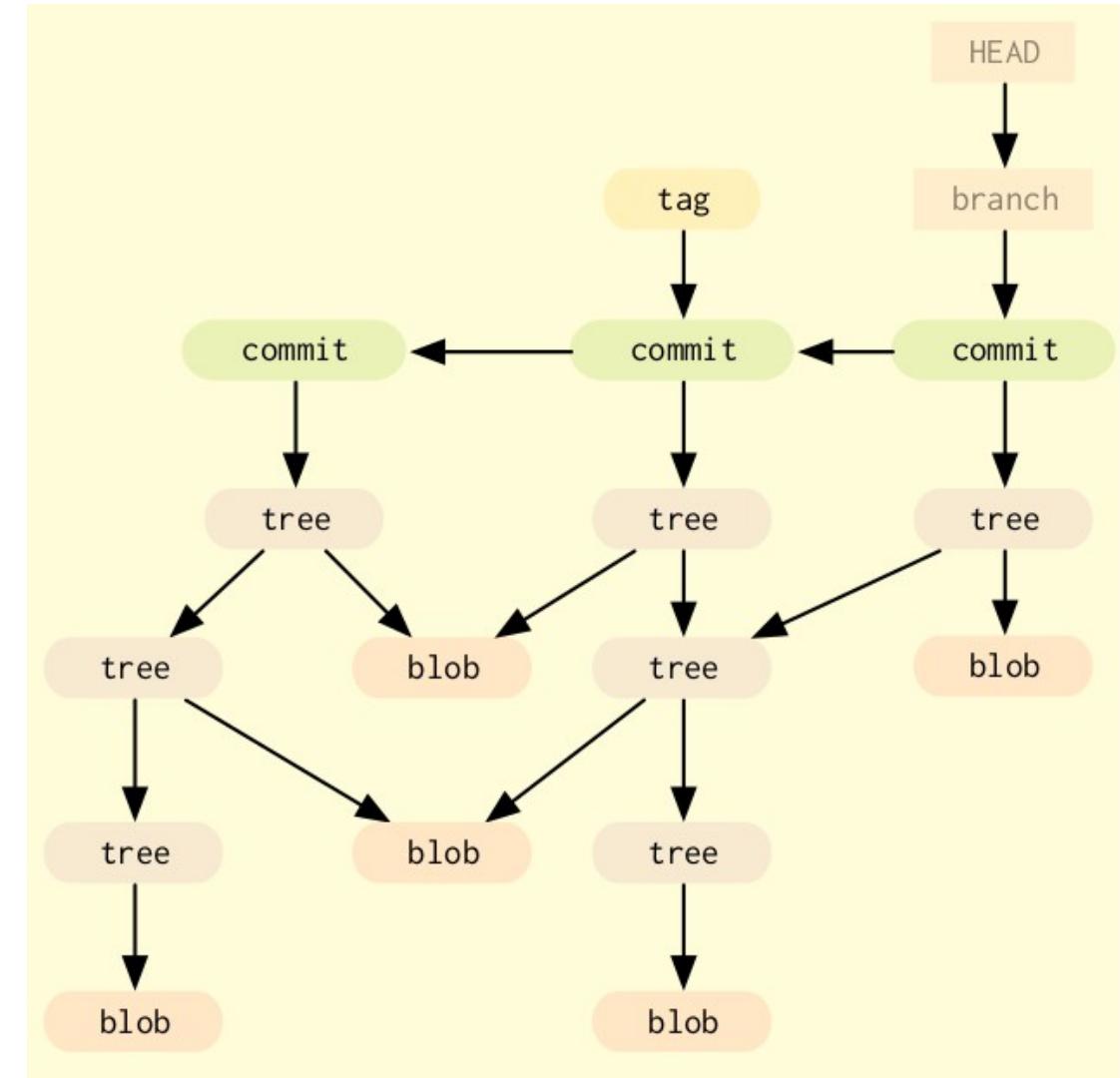
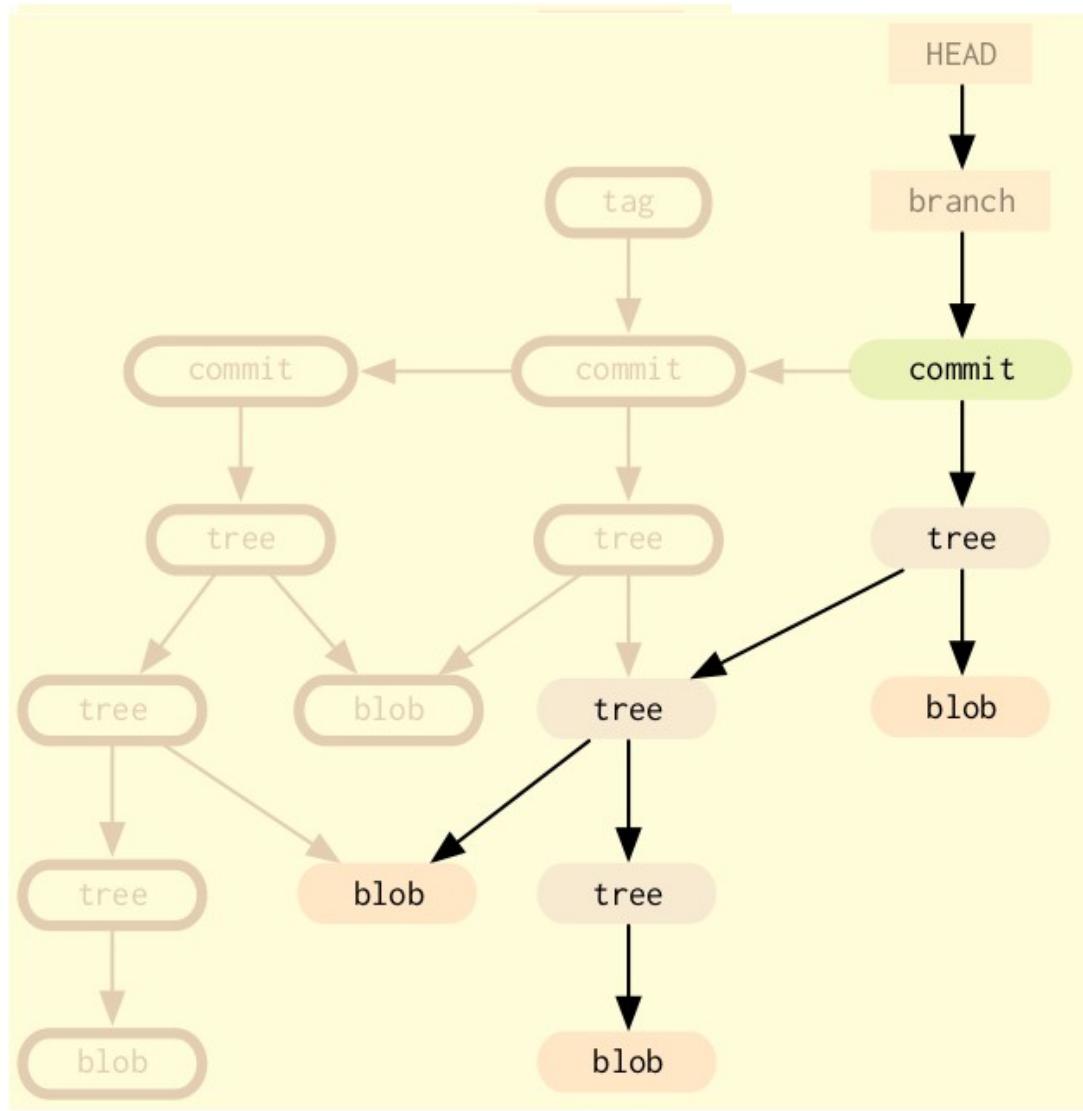


I tipi di oggetti Git: i riferimenti



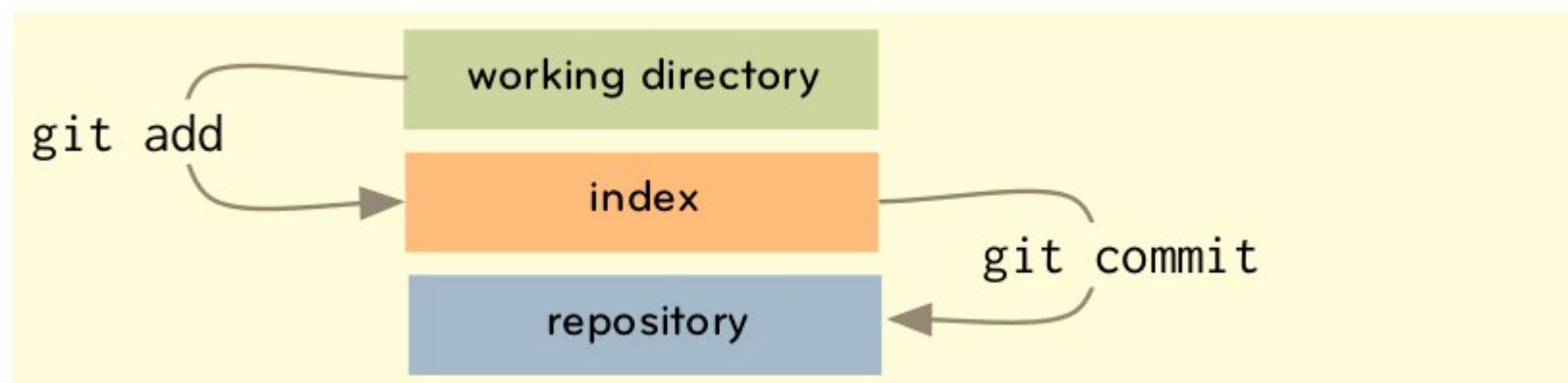
- I riferimenti
 - Sono puntatori a commit o ad altri riferimenti
- Esempi:
 - HEAD
 - I branch
 - master
 - remote ...
- Sono contenuti in `.git/refs`
- I riferimenti cambiano costantemente
 - ad esempio un riferimento di branch avanza ad ogni commit su quel branch

Tracciamento cambiamenti



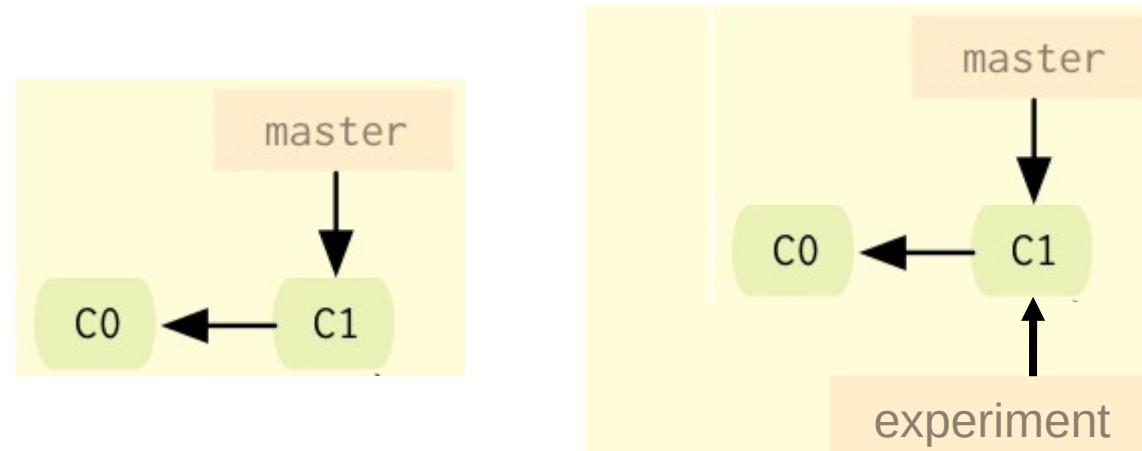
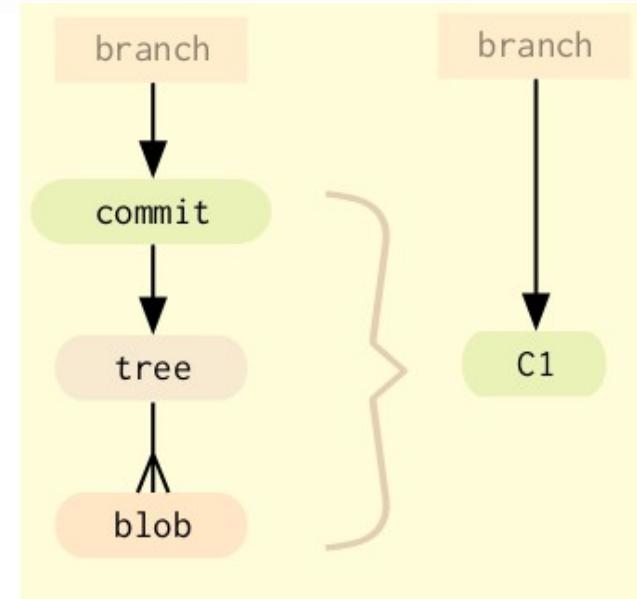


- La working directory
 - In sincrono con il file system locale e tiene traccia delle modifiche ai suoi file e directory
- Index/staging index/staging area
 - Tiene traccia delle modifiche inserite con `git add`, da memorizzare nel prossimo commit
- Commit history in repository
 - Il comando `git commit` inserisce i cambiamenti della staging area nella commit history
- Se si vogliono rendere permanenti le modifiche bisogna ricordarsi sempre di:
 - inserirle nella staging area
 - fare il commit

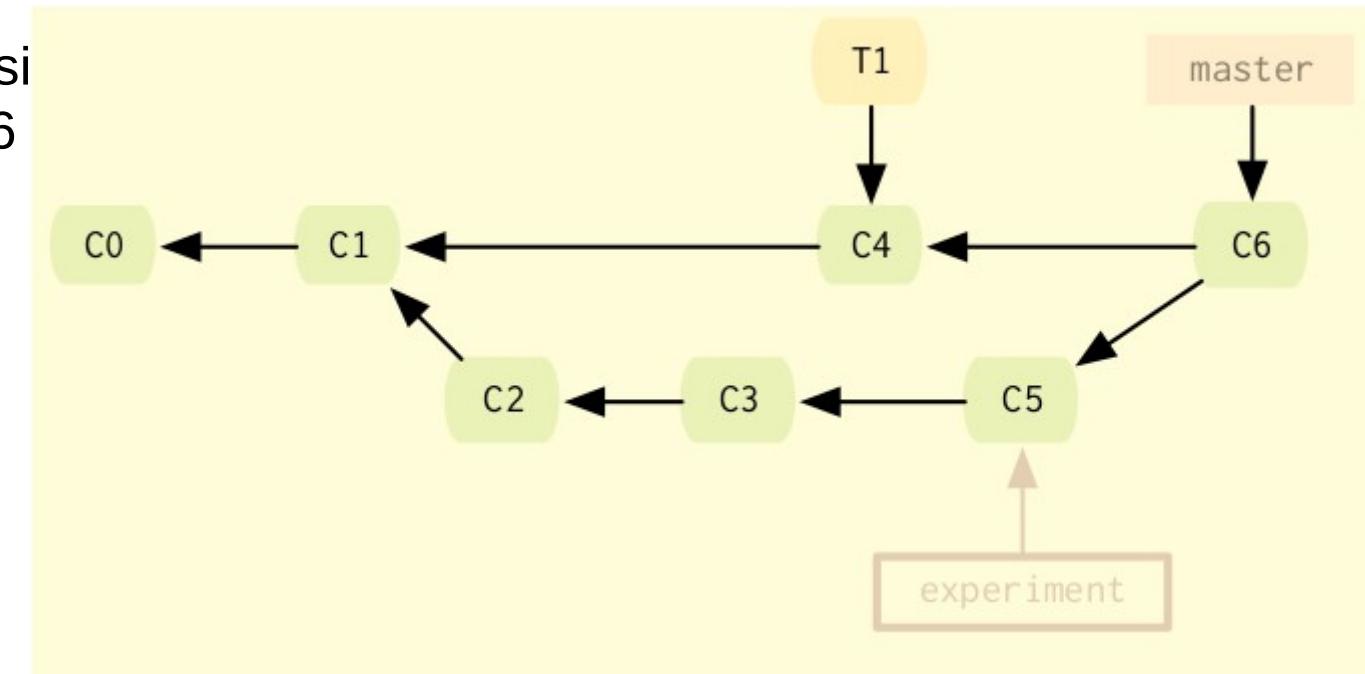


Git - branch e merge

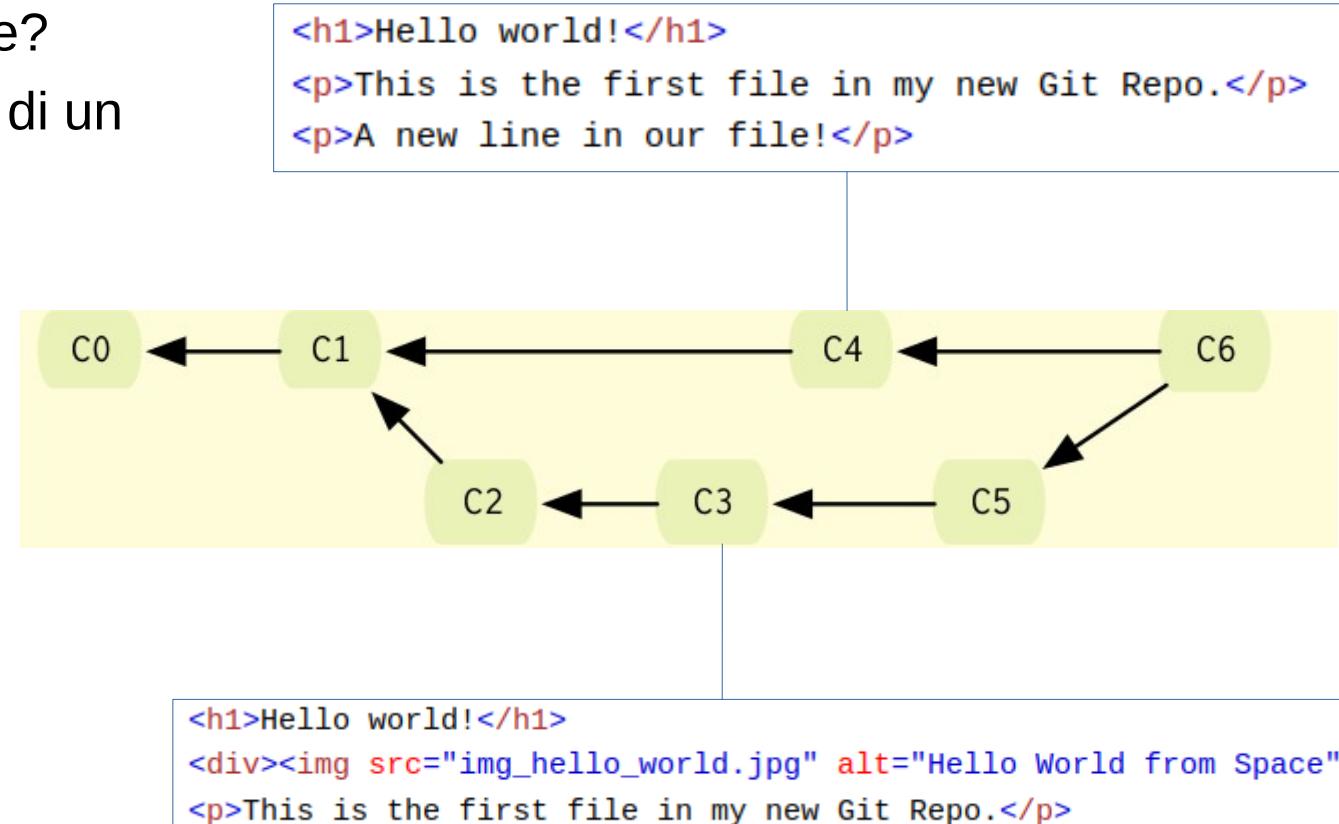
- Focalizziamoci ora sui commit
 - Ignoriamo la rappresentazione ad albero
- Consideriamo una possibile evoluzione:
- Nel branch master sono al commit C1
- bisogna realizzare una funzionalità sperimentale
- decido di creare un branch experiment
- Inizio a lavorare su experiment



- Nel frattempo sul master è proseguito il lavoro con due commit, l'ultimo etichettato con T1
- Sul ramo experiment ho ottenuto una versione stabile e testata della nuova funzione al commit C5
- Il team concorda di integrare la funzione nel master perciò si fa un merge:
`$ git merge experiment`
- Tutte le modifiche fatte da quando i branch si sono divisi vengono integrate nel commit C6 del master



- Cosa succede se commit in branch diversi contengono versioni diverse dello stesso file?
 - L'operazione di merge segnala la presenza di un **conflitto** da risolvere
-
- Bisogna quindi indicare la versione corretta da inserire nella staging area
 - si modifica il/i file causa del conflitto
 - ... e poi si fa il commit





Usare Git - creare/clonare repository

Creare una nuova directory in Git:

```
$ mkdir myproject
```

Entrare nella directory: \$ cd myproject

inizializzare il repository Git nella cartella

```
$ git init .
```

Initialized empty Git repository
in /home/davide/myproject/.git/

Creare un file vuoto:

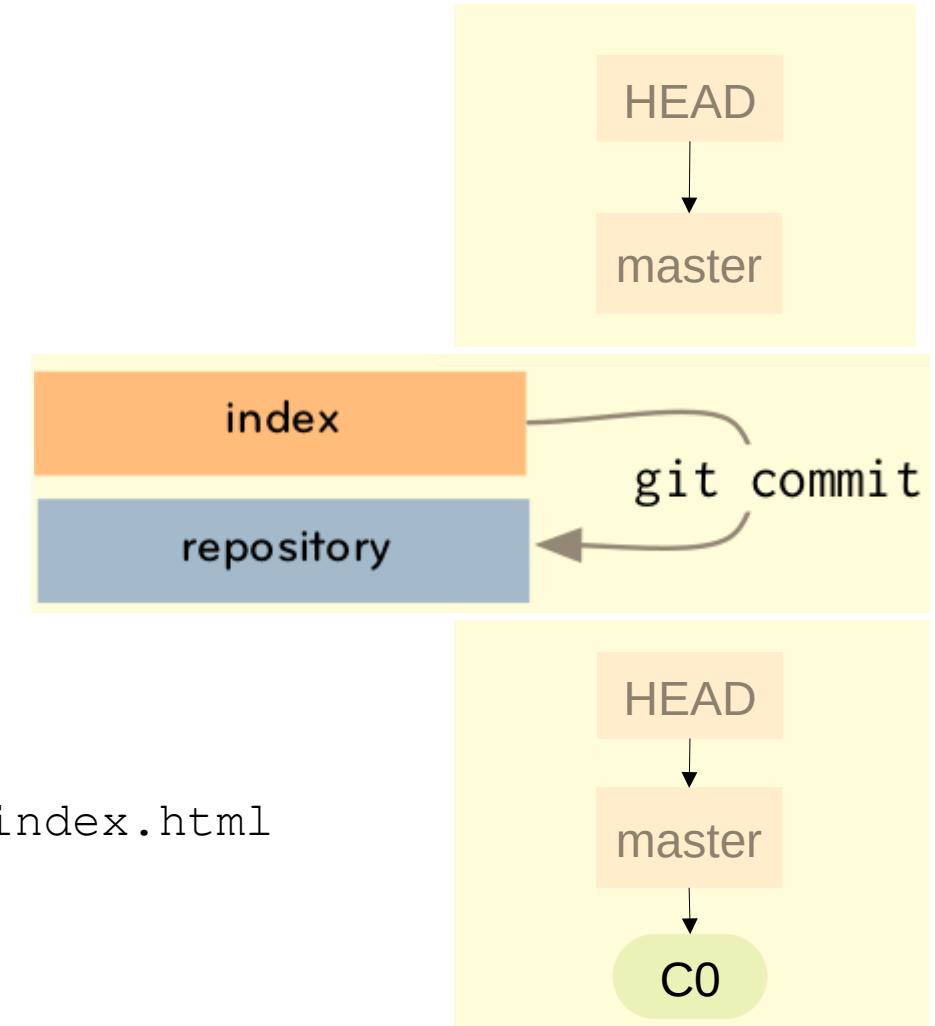
```
$ touch index.html
```

Aggiungere il file a quelli inclusi nel repository in Git:

```
$ git add index.html
```

Salvare la prima versione del file nel repository in Git:

```
$ git commit -m "creato file vuoto index.html"  
[master (root-commit) 3d382da] creato file vuoto index.html  
1 file changed, 0 insertions(+), 0 deletions(-)  
create mode 100644 index.html
```



Aprire README.md in un editor di testo, scrivere Hello Git e salvare il file

Visualizzare le modifiche tra l'ultima versione del repository e quella attualmente sul computer in Git:

```
$ git diff
diff --git a/README.md b/README.md
index e69de29..9f4d96d 100644
--- a/README.md
+++ b/README.md
@@ -0,0 +1 @@
+Hello Git
```

Esaminare stato del repository Git nella cartella

```
$ git status
Sul branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
    modified:   README.md
no changes added to commit (use "git add" and/or "git commit -a")
```

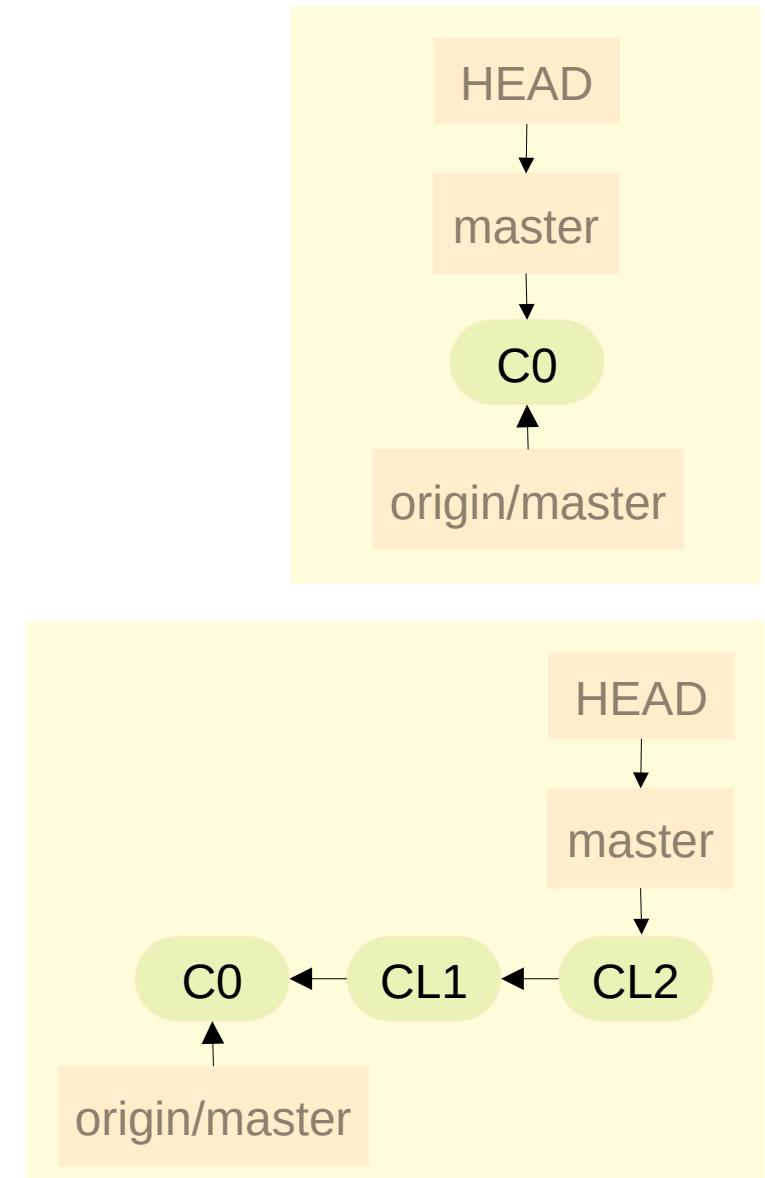
Altro modo per ottenere un repository è copiarne uno remoto:

```
$ git clone <url_rep_remoto>
```

Avremo due referenze:

- master, il branch principale locale
- origin/master, il branch principale remoto

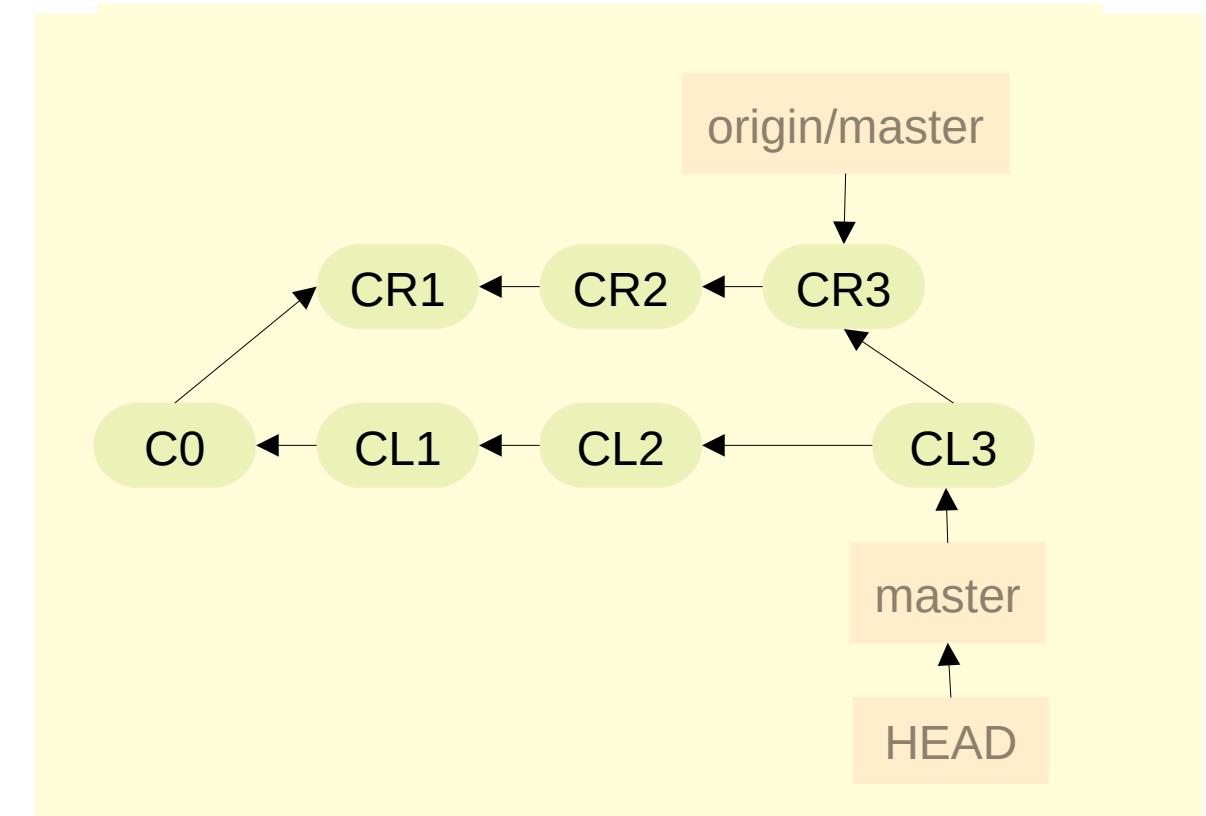
Se si lavora sul repository locale e si fanno cambiamenti cambiano i riferimenti locali, non quelli remoti



I cambiamenti nel repository locale si ricevono con:

```
$ git fetch
```

- Solo con un successivo `merge` i cambiamenti nel repository remoto saranno integrati con quello locale
- L'operazione di `pull` è equivalente a:
 - `pull = fetch + merge`



Si crea un nuovo branch con

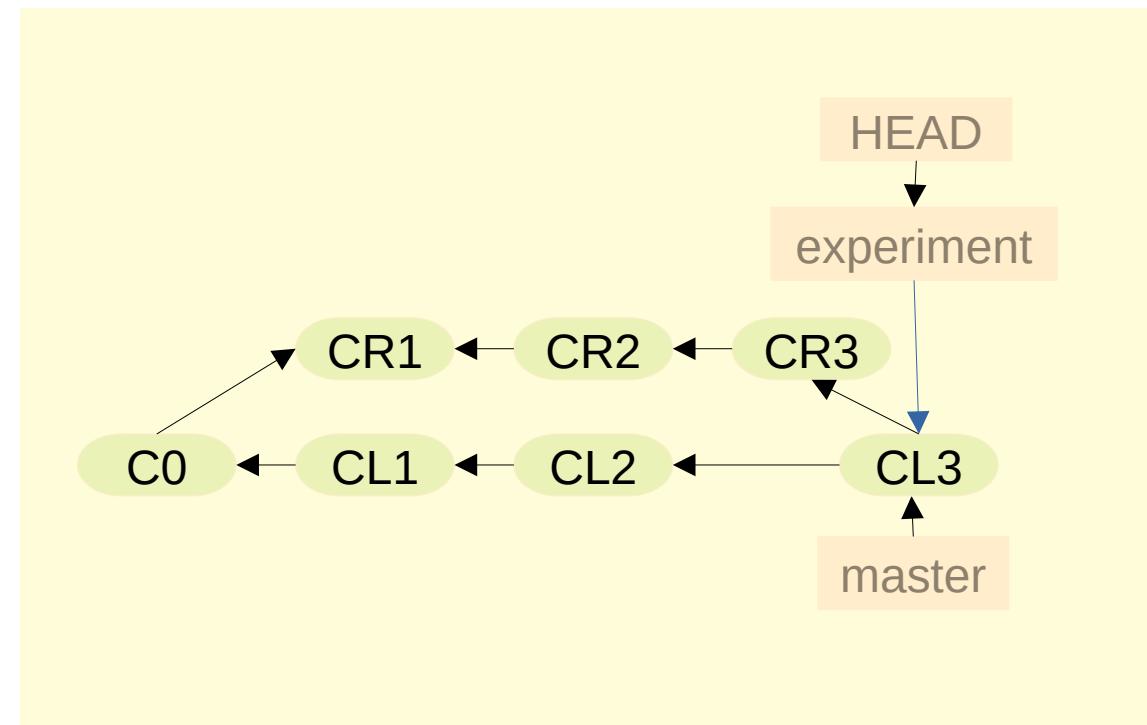
```
$ git branch <nome branch>
```

Si passa al nuovo branch tramite

```
$ git checkout <nome branch>
```

Si possono fare entrambe le operazioni con

```
$ git checkout -b <nome branch>
```



- Il checkout muove il puntatore HEAD ad un branch o ad uno specifico commit
- Reset permette di ripristinare tutto (stage area, work direcory, history) allo stato di un commit
- Revert di un commit crea un nuovo commit che inverte il precedente

Command	Scope	Common use cases
<code>git reset</code>	Commit-level	Discard commits in a private branch or throw away uncommitted changes
<code>git reset</code>	File-level	Unstage a file
<code>git checkout</code>	Commit-level	Switch between branches or inspect old snapshots
<code>git checkout</code>	File-level	Discard changes in the working directory
<code>git revert</code>	Commit-level	Undo commits in a public branch

Autenticazione e autorizzazione con Oauth2

Uso di JSON Web Token per gestire l'accesso alle risorse

Sommario

- Il protocollo OAUTH2
- JSON Web Token
- Un esempio di applicazione

Il protocollo OAUTH2



Oauth2 <https://oauth.net/2/> è un protocollo standard per gestire l'autorizzazione ad accedere a determinate risorse da parte di applicazioni web. Il protocollo è stato sviluppato dal IETF Working Group Oauth (RFC 6749)

Esso definisce un meccanismo per permettere ad un utente di fornire ad un'applicazione web il permesso di accedere (in modo controllato) a proprie risorse che risiedono su un server in rete.

L'applicazione web ottiene l'autorizzazione per il tramite di un Authorization Server che gestisce l'autenticazione dell'utente e fornisce all'applicazione web la CAPABILITY di accedere alle risorse concesse dall'utente nella forma di un TOKEN (con scadenza).

Già visto ... ?

Questo meccanismo è molto utilizzato e lo sperimentiamo (a nostra insaputa) tutte le volte che qualche servizio ci propone di registrarci attraverso le nostre credenziali Google, Facebook, ecc.

(per approfondire:<https://developers.google.com/identity/protocols/oauth2>)

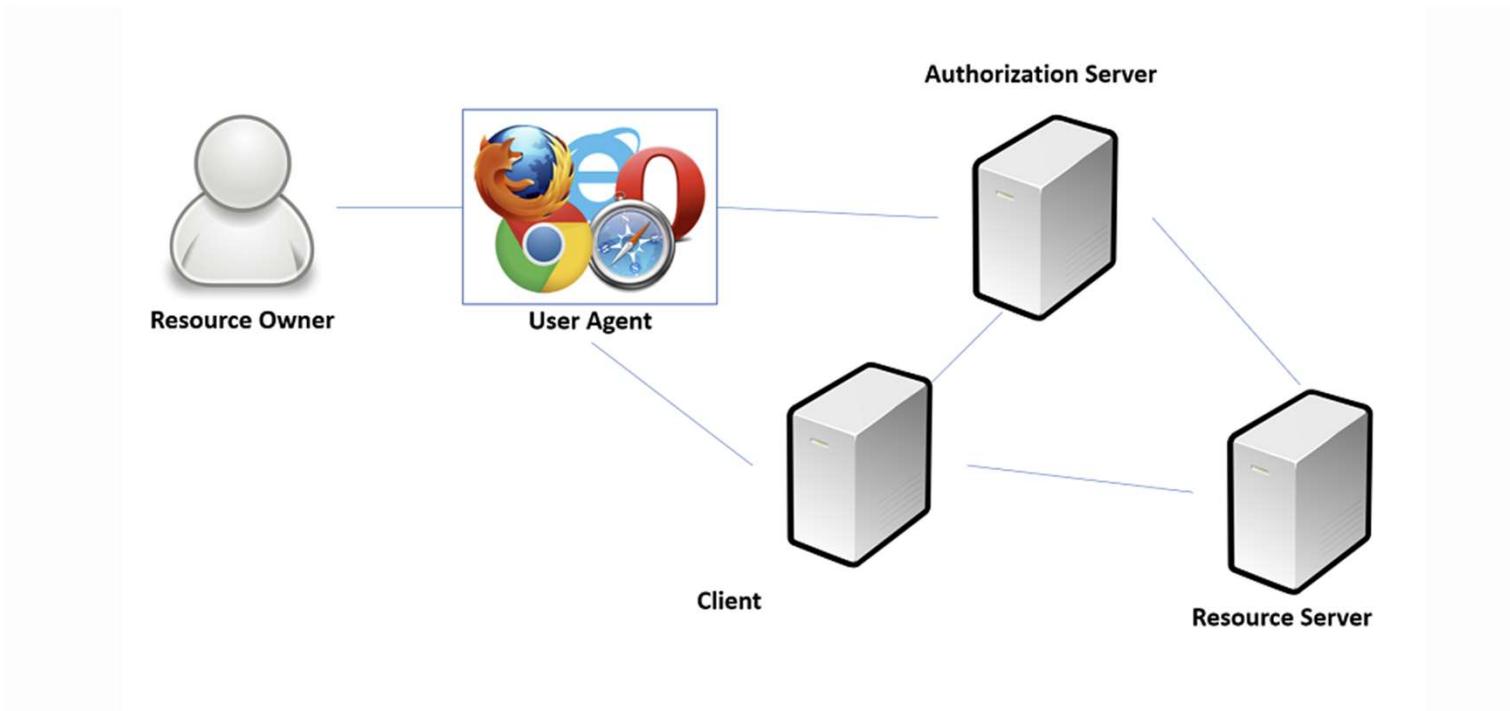
In questo caso l'interazione procede in questo modo: veniamo indirizzati sul servizio di autenticazione scelto (es. Google), qui inseriamo le credenziali di tale servizio (che la nuova applicazione NON potrà vedere), contestualmente ci viene chiesto se siamo disposti a condividere alcune informazioni con la nuova applicazione: per esempio l'indirizzo e-mail, e in qualche caso altre informazioni (per esempio l'età, i contatti, ecc.). Nel momento in cui accettiamo, l'applicazione acquisisce un TOKEN con il quale potrà accedere alle risorse che le abbiamo concesso (e solo a quelle) per un periodo di tempo limitato. Trascorso questo periodo avrà bisogno di aggiornare il Token per poter continuare ad accedere.

Terminologia

Per spiegare il funzionamento di Oauth2 è necessario introdurre alcuni termini, corrispondenti ai RUOLI coinvolti nel protocollo:

- Authorization server
- Resource server (gestore delle risorse dell'utente)
- User (resource owner) – che può utilizzare uno User Agent (browser)
- Client application (applicazione web che ha bisogno di poter accedere alle risorse dello User per offrirgli un certo servizio: per questo lo User dovrà concederle una capability per accedere – entro certi limiti – alle risorse necessarie)

Terminologia



Nel progetto per la distribuzione delle risorse idriche ...

Per contestualizzare:

- l'utente (agricoltore o fornitore acqua) interagisce con un'applicazione web che corrisponde al ruolo di *Client Application*,
- il servizio che mantiene i dati corrisponde al ruolo del *Resource Server*,
- l'Authorization server è un servizio esterno (vedremo alcuni esempi: Keycloak, ma anche il server Fitbit) che si occupa di autenticare un utente, e successivamente fornisce alla *Client Application* un Token (con scadenza temporale definita) che questa potrà utilizzare per richiedere al *Resource Server* l'accesso (parziale) alle risorse dell'utente, in base a quanto indicato nel Token.

Tipi di Client Application: Confidential vs Public

Il protocollo prevede diverse forme di acquisizione dei permessi di accesso; ciò è motivato da un'esigenza di flessibilità: deve poter funzionare in diversi casi:

1. La client application è su un server protetto da possibili manipolazioni (e viene quindi classificata come CONFIDENTIAL)
2. La client application è una Single Page Application, costituita da codice javascript, che viene eseguito direttamente nel browser: quindi è potenzialmente attaccabile (in questo caso è classificata come PUBLIC)

Authorization Grant

Lo standard definisce diversi tipi di Authorization Grant. Noi vedremo in particolare quello denominato AUTHORIZATION CODE

Altri tipi, che non vedremo, sono:

- Proof Key of Code Exchange (PKCE)
- Client Credentials
- Device Code
- Implicit (ora sconsigliato e sostituito da PKCE)

Authorization code

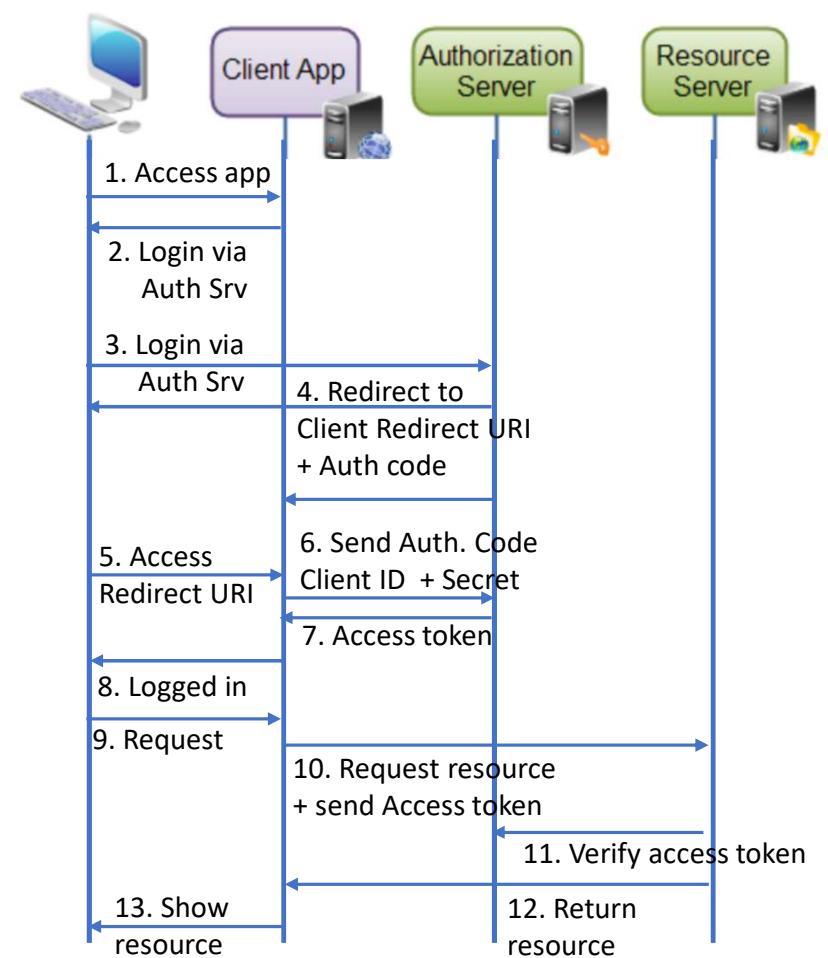
Prima di esaminare il funzionamento del protocollo per questo tipo di authorization grant occorre fare una premessa:

- la Client Application con cui l’utente interagisce per accedere al Resource Server è PREVENTIVAMENTE REGISTRATA presso l’Authorization Server. Nel caso tale applicazione sia considerata «CONFIDENTIAL» al momento della registrazione acquisisce un «SECRET» che si suppone possa proteggere da accessi indesiderati.

Nota: l’ultima assunzione (possesso del «SECRET») non è indispensabile, ma aggiunge un livello di sicurezza ulteriore.

Il protocollo

1. Il proprietario delle risorse (user) accede all'applicazione client.
2. L'applicazione client rinvia l'utente al login presso l'auth. server.
3. Per il login l'utente è ridiretto all'authorization server.
4. Quindi l'utente si autentica con l'authorization server. Se autenticato gli viene chiesto se intende concedere le risorse richieste all'applicazione. Se l'utente conferma è rinviato alla applicazione client (via Client Redirect URI) e con un Auth.Code
5. Il rinvio avviene mandando l'utente alla "redirect URI", specificato all'atto della registrazione. Insieme al rinvio, l'authorization server manda un codice che rappresenta l'autorizzazione (Auth. Code)
6. Quando si accede all'URI nell'applicazione client questa si connette direttamente all'authorization server. E invia il codice di autorizzazione, il client ID (ed eventualmente il segreto)
7. Se l'authorization server li ritiene validi ritorna un *token d'accesso*.
8. Lo user riceve conferma e può iniziare a fare le sue richieste alla client application (9)
10. L'applicazione client può ora usare il token per chiedere accesso alle risorse al resource server. Il token vale sia come autenticazione del client e del proprietario delle risorse (user), che come diritto di accesso alle risorse.
11. Il resource server può verificare la validità del token, in caso positivo risponde restituendo la risorsa (12, 13)



JSON Web Token (1/2)

- Il token usato per l'accesso può assumere diversi formati, uno dei più diffusi è il JWT – JSON Web Token
(per approfondimenti vedere jwt.io dove sono anche indicate varie librerie per la decodifica dei token, l'estrazione delle informazioni contenute nonché la verifica dell'autenticità del token)
- Il JWT comprende tre parti (codificate in BASE64 – base64url3):
 - Header
 - Payload (che potrà contenere varie informazioni, tra cui il tipo di permessi di accesso concessi con questo token)
 - Signature/Encryption data
- Il Resource Server può controllare l'autenticità del token verificando che il token sia firmato dall'Authorization Server (tramite la sua chiave pubblica).

JSON Web Token (2/2)

ESEMPIO:

```
eyJhbGciOiJIUzI1NilsInR5cCI6IkpXVCJ9.eyJzdWliOilxMjM0NTY3ODkwliwib  
mFtZSI6Ikpvag4gRG9IiwiWF0ljoxNTE2MjM5MDlyfQ.SflKxwRJSMeKKF2  
QT4fwpMeJf36POk6yJV_adQssw5c
```

DECODIFICATI:

Header: { "alg": "HS256", "typ": "JWT" }	Payload: { "sub": "1234567890", "name": "John Doe", "iat": 1516239022 }
---	---

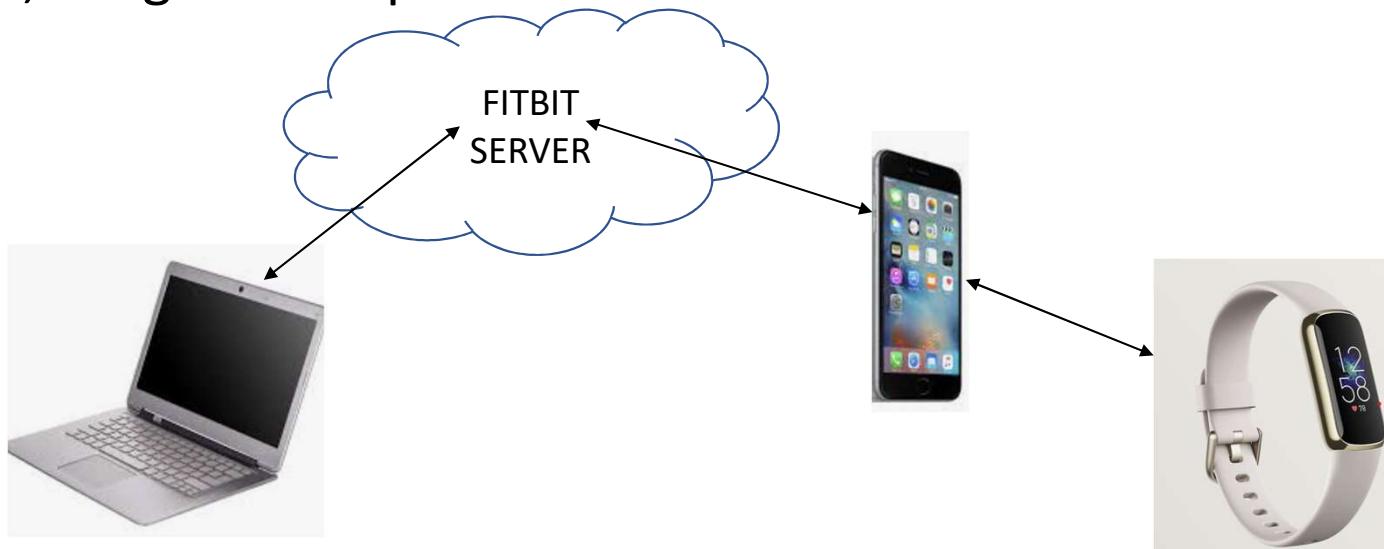
Collegandosi a jwt.io e inserendo il JWT sopra, potete verificarne l'autenticità inserendo la parola *secret* nel campo Verify Signature.

Autenticazione e autorizzazione con Oauth2 un esempio d'uso: FITBIT

Uso di JSON Web Token per gestire l'accesso alle risorse

Un esempio di applicazione: accedere ai dati del profilo fitbit

- La fitness band Fitbit si collega via bluetooth al cellulare, ma le informazioni vengono mandate con una certa periodicità al cloud di fitbit. Le informazioni pur essendo presenti nella fitness band e sul cellulare, vengono sempre lette dal server su cloud



Lettura dei dati dell'utente da programma

Si possono sviluppare applicazioni che si connettono al server fitbit utilizzando le API che mette a disposizione

<https://dev.fitbit.com/build/reference/web-api>

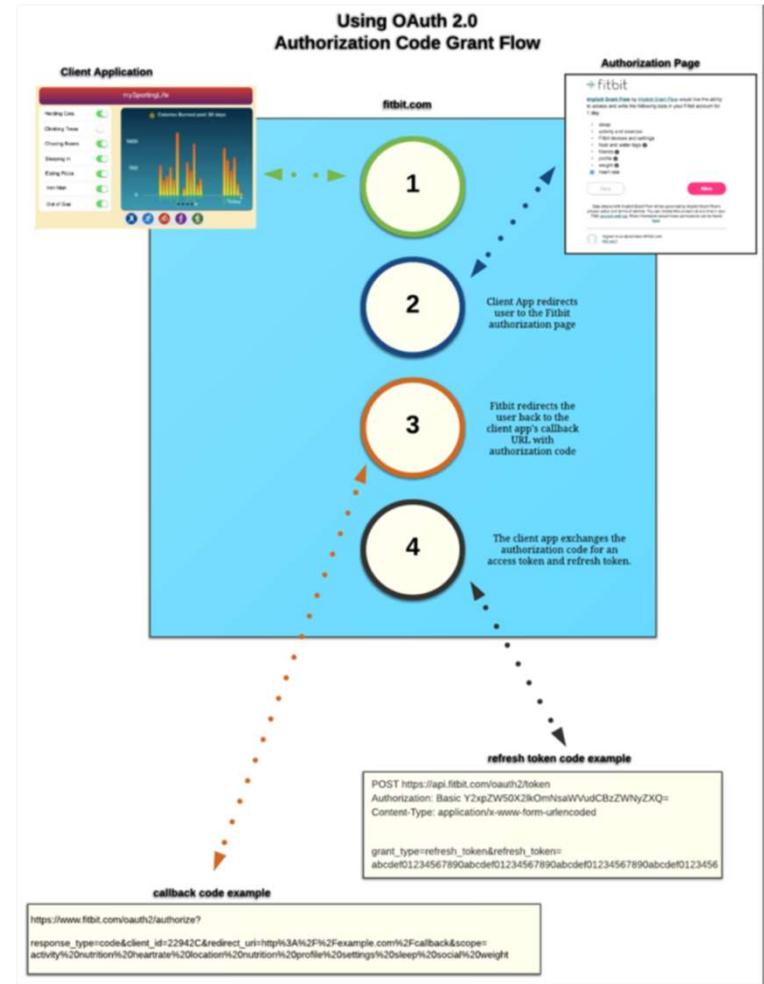
Quick Start

1. [Review the basics](#) about how the Fitbit Web API works.
2. [Register your application](#) to get API client credentials. You will need a Fitbit account (free) to register an app.
3. Implement an [OAuth 2.0 authorization flow](#) to allow people to give your app permission to access data on their behalf.
4. Make HTTP requests to access data. The different types of data available via the Web API are listed in the navigation. You can also try the [API Explorer](#).
5. If you have a server app and want to be notified when people have new data available, implement the [Subscriptions API](#).

The Authorization Code Grant Flow has the following steps:

1. Your application presents the Fitbit authorization page to the user by calling the "authorize" endpoint. See [Authorization Page](#) below.
2. The user consents to share their Fitbit data with your application by enabling some or all scopes. Upon the user's initial consent, Fitbit redirects the user back to your application using the redirect or callback URL with an authorization code as a URL parameter.
3. Your application exchanges the authorization code it receives from step 2 for an access token and refresh token pair. See [Access Token Request](#) below. Your application should store the access token and refresh token.
4. The application uses the access token to execute API calls. The refresh token is used to renew the access token when it expires without having to re-prompt the user.

Dal sito <https://dev.fitbit.com/build/reference/web-api/oauth2/#authorization-code-grant-flow>



Prima di registrare la nuova app (la client application) ...

Occorre disporre di un account su fitbit.com (potete crearne uno locale oppure registrarvi con un utente google). Una volta creato l'account potete accedere alla vostra dashboard dove sono visualizzabili i dati provenienti dal vostro fitness tracker/smartwatch o dati inseriti manualmente.

The screenshot shows the Fitbit website interface. At the top, there's a navigation bar with links for 'Pannello', 'Diario', 'Community', and 'STORE'. A banner at the top right asks if you want to configure a new device. Below the banner, the dashboard displays various activity metrics: 'Attività recente' (Swimming, Walking), 'Peso' (Weight goal set), and a calorie counter showing 1.559 calories. On the left, there's a graph for 'Passi' (Steps) and a section for 'Registra attività' (Record activity) with a red box highlighting it. The 'Attività recente' section shows:

Atividade	Data	Durata	Calorie
Nuoto	11 mag.	51 minuti	614 cal.
Camminata	11 mag.	1/0 minuti	1/0 cal.
Nuoto	10 mag.	51 minuti	444 cal.

The 'Registra attività' form includes fields for 'Atividade comuni' (Walking, Running, Swimming, Biking), 'Camminata' (Walking), 'Data' (11-05-2021), 'Ora di inizio' (Start time), 'Durata' (Duration), and 'Distanza' (Distance).

Le Web-API di fitbit.com

Invieremo al server fitbit.com delle GET per prelevare le attività dell’utente [user-id] ad una certa data (nel formato AAAA-MM-GG)

Dovremo inviare nell’header un token dopo averlo prelevato dall’authorization server di fitbit (presentando l’ID di una Client App registrata sullo stesso, e un «secret» condiviso tra Client App e l’authorization server)

Activity & Exercise Logs

Get Daily Activity Summary

The **Get Daily Activity Summary** endpoint retrieves a summary and list of a user’s activities and activity log entries for a given day in the format requested using units in the unit system which corresponds to the Accept-Language header provided.

Privacy Setting

The **Activities (Friends or Anyone)** privacy permission grants access to a user’s resource with the exception that the response **DOES NOT** include a detailed list of activity log entries.

Considerations

1. Daily summary data and daily goals for elevation (**elevation**, **floors**) only included for users with a device with an altimeter.
2. The **steps** field in activity log entries included only for activities that have steps (e.g. "Walking", "Running"); **distance** only included when it is relevant.
3. Calorie burn goal (**caloriesOut**) represents either dynamic daily target from the premium trainer plan or manual calorie burn goal. Goals are included to the response only for today and 21 days in the past.

Resource URL

GET [https://api.fitbit.com/1/user/\[user-id\]/activities/date/\[date\].json](https://api.fitbit.com/1/user/[user-id]/activities/date/[date].json)

user-id	The encoded ID of the user. Use “.” (dash) for current logged-in user.
date	The date in the format <code>yyyy-MM-dd</code>

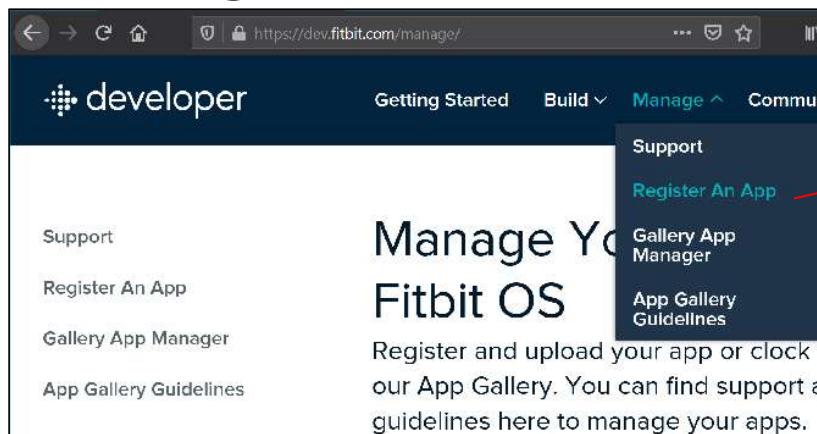
Request Headers

Accept-Locale	optional	The locale to use for response values.
Accept-Language	optional	The measurement unit system to use for response values.

Example Response

```
{  
  "activities": [  
    {  
      "activityId": 51007,  
      "activityParentId": 90019,  
      "calories": 230,  
      "description": "7mph",  
      "distance": 2.04,  
      "duration": 1097053,  
      "hasStartTime": true,  
      "isFavorite": true,  
      "logId": 1154701,  
      "name": "Treadmill, 0% Incline",  
      "startTime": "00:25",  
      "steps": 3783  
    }  
,  
    "goals": {  
      "caloriesOut": 2826,  
      "distance": 8.05,  
      "steps": 10000  
    }  
  ]  
}
```

Registriamo una nuova app ([dev.fitbit.com](https://dev.fitbit.com/manage/))



The screenshot shows the Fitbit developer portal interface. At the top, there's a navigation bar with links for 'Getting Started', 'Build', 'Manage', and 'Community'. A dropdown menu under 'Manage' is open, showing options like 'Support', 'Register An App' (which is highlighted with a red arrow), 'Gallery App Manager', and 'App Gallery Guidelines'. Below this, the main content area has a title 'Manage Your Apps' and a sub-section 'Fitbit OS'. It contains instructions: 'Register and upload your app or clock face to our App Gallery. You can find support and guidelines here to manage your apps.' On the left side, there's a sidebar with links for 'Support', 'Register An App', 'Gallery App Manager', and 'App Gallery Guidelines'.

Application name / Description: a vostra scelta

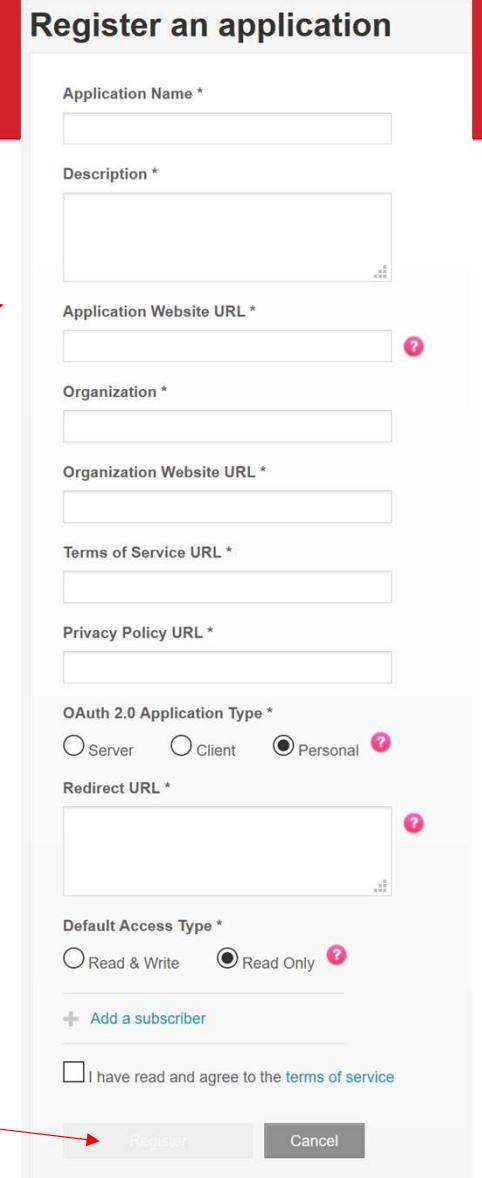
Application Website, Terms of Service, Privacy policy URL: <http://localhost>

Organization + Website URL: a vostra scelta (io ho inserito DISIT, UPO + sito)

Oauth2 Application Type: Personal (potrà accedere solo l'utente che ha creato l'app)

Redirect URL: <http://localhost:6789/Callback>

Access Type: Read Only; flaggare «I have read ...» poi Click su Register



The screenshot shows a 'Register an application' form. It includes fields for 'Application Name *', 'Description *', 'Application Website URL *', 'Organization *', 'Organization Website URL *', 'Terms of Service URL *', 'Privacy Policy URL *', 'OAuth 2.0 Application Type *' (with 'Personal' selected), 'Redirect URL *', 'Default Access Type *' (with 'Read Only' selected), and a checkbox for 'I have read and agree to the terms of service'. At the bottom, there are 'Register' and 'Cancel' buttons.

Annotare i campi:

- Client ID
- Client Secret
- Redirect URL (definita da noi)
- Oauth2.0 Authorization URI

<https://www.fitbit.com/oauth2/authorize>

- Oauth 2.0 Access/Refresh Token Request URI

<https://www.fitbit.com/oauth2/token>

The screenshot shows a user interface for managing OAuth applications. At the top, there's a navigation bar with tabs: 'OVERVIEW', 'REGISTER AN APP', and 'MANAGE MY APPS'. Below this, a section titled 'Applications I registered' lists a single application named 'UnaAppdiProva'. This application is described as 'Prova libit'. There are four buttons below the application name: 'Edit Application Settings' (pink), 'Delete Application' (pink), 'Reset Client Secret' (pink), and 'Revoke Client Access Tokens' (pink). The application details are listed in five boxes:

- OAuth 2.0 Client ID:** 239YVM
- Client Secret:** 98a8baf5895d19de0b2d5e0bf0929dc4
- Redirect URL:** http://localhost:6789/Callback
- OAuth 2.0: Authorization URI:** https://www.fitbit.com/oauth2/authorize
- OAuth 2.0: Access/Refresh Token Request URI:** https://api.fitbit.com/oauth2/token

A link 'OAuth 2.0 tutorial page' is visible at the bottom of the application details section.

Sperimentiamo (vedere ... fitbit-oauth-java sul dir)

Utilizziamo le librerie di google che implementano il protocollo Oauth2 con «accessori» per avviare un web-server jetty quando si esegue l'applicazione, e per parsificare il json ricevuto dal server.

In **build.gradle**

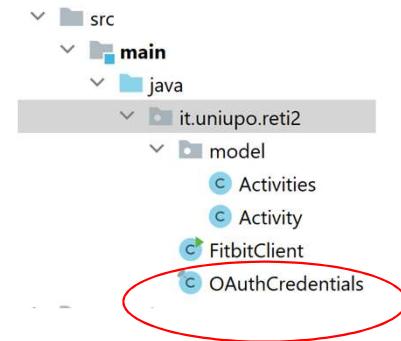
```
dependencies {  
    ....  
    // OAuth client library  
    implementation 'com.google.oauth-client:google-oauth-client:1.28.0'  
    // Jetty extension for the OAuth client library (as local webserver)  
    implementation 'com.google.oauth-client:google-oauth-client-jetty:1.28.0'  
    // gson support for the OAuth client library  
    implementation 'com.google.http-client:google-http-client-gson:1.28.0'  
}
```

Il programma (un web server che risponderà su *localhost:6789*)

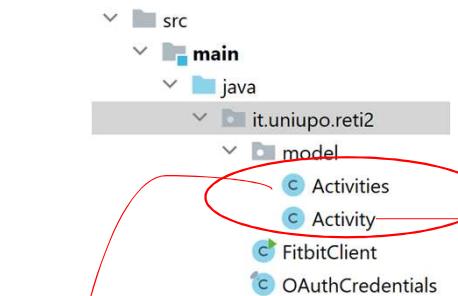
Tutti i dati ricavati al momento della registrazione della «client application» sul sito fitbit si devono inserire in **OauthCredentials.java**

```
// sample client id, as provided by Fitbit
private static final String CLIENT_ID = "239YWF";
// sample client secret, as provided by Fitbit
private static final String CLIENT_SECRET = "821b9097c207ffa914c985a8a51bb8a3";
// domain and port of the local server to complete the OAuth 2.0 authorization flow
private static final int PORT = 6789;
private static final String DOMAIN = "localhost";

// server URLs, as provided by Fitbit
private static final String TOKEN_SERVER_URL = "https://api.fitbit.com/oauth2/token";
private static final String AUTHORIZATION_SERVER_URL = "https://www.fitbit.com/oauth2/authorize";
```



Il modello dei dati (per estrarre i dati dal json ottenuto dal server fitbit)



<https://dev.fitbit.com/build/reference/web-api/activity/>

```
public class Activities {  
  
    // one of the objects in the "activities" JSON  
    private ArrayList<Activity> activities;  
  
    /* Getters */  
    public ArrayList<Activity> getActivities() { return activities; }  
}
```

```
/**  
 * Represents an activity, according to  
 * the Fitbit Web API  
 */  
public class Activity {  
    // private String name  
    private String activityParentName;  
    // calories  
    private int calories;  
    // activity description  
    private String description;  
    // distance (km)  
    private float distance;  
    // steps  
    private int steps;  
    ...  
    ... }  
}
```

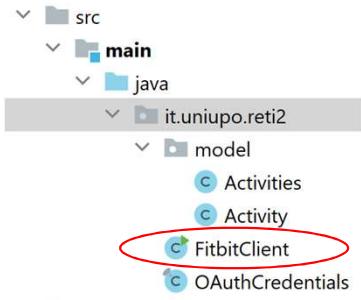
The JSON response from the Fitbit server is shown on the right, with arrows pointing from specific fields in the Java code to their corresponding JSON keys. A red arrow points from the `Activities` class in the code to the `"activities": [` part of the JSON. Another red arrow points from the `Activity` class in the code to the `Activity` object in the JSON. Blue arrows point from the `activityParentName`, `calories`, `description`, `distance`, and `steps` fields in the Java code to their respective JSON keys. A red arrow also points from the `getActivities()` method in the code to the `"activities": [` part of the JSON.

array di attività

"activities": [
 {
 "activityId": 51007,
 "activityParentId": 90019,
 "activityParentName": "Run",
 "calories": 230,
 "description": "7mph",
 "distance": 2.04,
 "duration": 1097053,
 "hasStartTime": true,
 "isFavorite": true,
 "logId": 1154701,
 "name": "Treadmill, 0% Incline",
 "startTime": "00:25",
 "steps": 3783
 }]

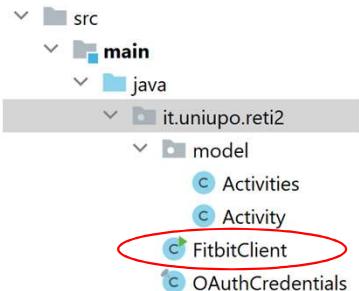
json restituito dal server quando interrogato dall'applicazione

Classe principale (fase di autorizzazione e interrogazione)



```
public static void main(String[] args) {
    try {
        // authorization flow
        final Credential credential = authorize();
        // print JWT access token
        System.out.println(credential.getAccessToken().toString());
        System.out.println("Authorization completed");
        // init a requestFactory
        HttpRequestFactory requestFactory =
            HTTP_TRANSPORT.createRequestFactory((HttpRequest request) -> {
                credential.initialize(request);
                request.setParser(new JsonObjectParser(JSON_FACTORY));
            });
        // execute the API call, backed by OAuth 2.0
        run(requestFactory);
        System.out.println("Authorized");
        // Success!
        return;
    } catch (IOException e) {
        System.err.println(e.getMessage());
    } catch (Throwable t) {
        t.printStackTrace();
    }
}
```

Classe principale (fase di autorizzazione e interrogazione)



```
private static void run(HttpRequestFactory requestFactory) throws IOException {
    // the URL to call
    GenericUrl url = new GenericUrl("https://api.fitbit.com/1/user/-/activities/date/2022-05-03.json");
    // perform a GET request
    HttpRequest request = requestFactory.buildGetRequest(url);

    // get the response as a JSON (and put it in a string)
    String jsonResponse = request.execute().parseAsString();

    // debug
    System.out.println("[DEBUG] " + jsonResponse);

    // parse the JSON string in POJO thanks to gson
    Activities activities = gson.fromJson(jsonResponse, Activities.class);

    // print out the steps and calories of the first activity (e.g., 1000 steps for a walk)
    for (int i = 0; i < activities.getActivities().size(); i++) {
        System.out.println("Name: " + activities.getActivities().get(i).getName());
        System.out.println("Description: " + activities.getActivities().get(i).getDescription());
        System.out.println("Steps: " + activities.getActivities().get(i).getSteps());
        System.out.println("Calories: " + activities.getActivities().get(i).getCalories());
    }
}
```

Richiesta relativa all'utente autenticato che deve anche essere il creatore dell'app

com.google.api.client.http.HttpRequestFactory
public com.google.api.client.http.HttpRequest buildGetRequest(@Nullable com.google.api.c...
 Builds a GET request for the given URL.
 Params: url – HTTP request URL or null for none
 Returns: new HTTP request
 Throws: java.io.IOException
 Inferred annotations: @org.jetbrains.annotations.Nullable
 Gradle: com.google.http-client:google-http-client:1.28.0

Il json viene parsificato secondo il modello e inserito in un'ArrayList

```
com.google.gson.Gson
public <T> T fromJson(@Nullable String json,
                      @NotNull Class<T> cClassOfT)
throws com.google.gson.JsonSyntaxException
```

This method deserializes the specified Json into an object of the specified class. It is not suitable to use if the specified class is a generic type since it will not have the generic type information because of the Type Erasure feature of Java.

Eseguiamo l'app sull'utente che ha svolto 3 attività

Cronologia attività

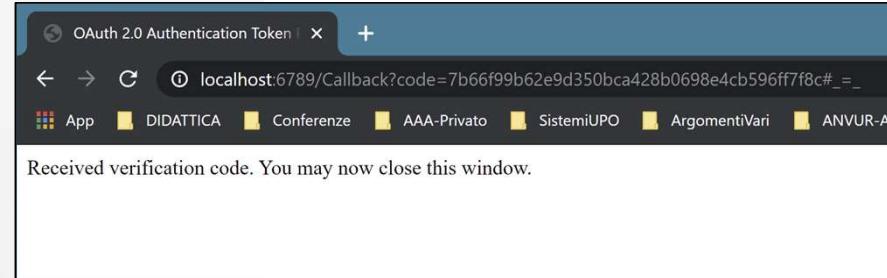
Data	Attività	Passi	Distanza	Durata	Calorie	
Oggi, 18:00	Corsa	6.878	6,5 chilometri	1:10:40	478 cal.	Mostra dettagli
Oggi, 13:00	Nuoto	N/D	3000 meters	1:30:55	614 cal.	Mostra dettagli
Oggi, 07:00	Camminata	7.095	5 chilometri	1:00:50	170 cal.	Mostra dettagli

Avvio della fase di autorizzazione

Viene automaticamente aperto il browser sulla pagina di autenticazione di fitbit, dove ci viene richiesto anche di scegliere QUALI dati mettere a disposizione della client-app.



The screenshot shows a browser window with the URL `accounts.fitbit.com/login?targetUrl=https%3A%2F%2Fwww.fitbit.com%2Flogin%2Ftransferpage%3Fredirect%3D...` highlighted by a red circle. The main content area displays the Fitbit login page with fields for EMAIL and PASSWORD, and a "Continua con Google" button.



The screenshot shows a browser window titled "OAuth 2.0 Authentication Token" with the URL `localhost:6789/Callback?code=7b66f99b62e9d350bca428b0698e4cb596ff7f8c#_=...`. It displays the message "Received verification code. You may now close this window."

Se si usa un account diverso da chi ha creato l'app viene bloccato:



The screenshot shows a browser window with the URL `fitbit.com/oauth2/authorize?client_id=23B4Z8&redirect_uri=http%3A%2F%2Flocalhost%3A6789%2FCallback&respo...` . It displays an error message from Fitbit stating: "L'app che stai tentando di connettere non ha fornito informazioni valide a Fitbit. Segnala questo problema al supporto clienti." and "Developer information: unauthorized_client - A "Personal" application is only authorized to request access tokens from the owner of the application."

La prima volta chiede anche di specificare quali informazioni si autorizza a prelevare



ProvaApp di Università del Piemonte Orientale desidera la possibilità di accedi ai seguenti dati nel tuo account Fitbit.

Avvertenza! Questa app non utilizza il protocollo HTTPS per ottenere in modo sicuro l'autorizzazione.

- Autorizza tutto
 - Dispositivi e impostazioni Fitbit
 - battito cardiaco
 - profilo 
 - diari alimenti e acqua 
 - amici 
 - attività e allenamento
 - peso 
 - sonno
 - posizione e GPS

Se autorizzi solo alcuni di questi dati, ProvaApp potrebbe non funzionare come previsto. Ulteriori informazioni su queste autorizzazioni [qui](#).

Nega

Consenti

I dati condivisi con ProvaApp saranno regolati da [Informativa sulla privacy](#) e [Termini del servizio](#) di Università del Piemonte Orientale. È possibile revocare il consenso in qualsiasi momento nel tuo [impostazioni account](#) Fitbit.



Risultato dell'esecuzione

Please open the following address in your browser: **Authorization endpoint**

https://www.fitbit.com/oauth2/authorize?client_id=239YWF&redirect_uri=http://localhost:6789/Callback&response_type=code&scope=activity%20heartrate%20location%20profile

Attempting to open that address in the default browser now...

2021-05-11 22:52:08.725:INFO::Stopped SocketConnector@localhost:6789

token

eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiIyMzLzV0YiLCJzdWIiOiI5RDc5S1EiLCJpc3MiOiJGaXRiaXQoILCJOeXAiOiJhY2Nlc3NfdG9rZW4iLCJyZ29wZXMiOiJyc29jIHZHjY3QgcNldCBybG9jIHZJ3ZWKqcmhB

Authorization completed

```
[DEBUG] {"activities":[{"activityId":17151,"activityParentId":90013,"activityParentName":"Walk","calories":170,"description":"Walking less than 2 mph, strolling"}]}
```

Name: Walk

Description: Walking less than 2 mph, strolling very slowly

Steps: 7095

Calories: 170

Name: Swim

Description: 25-50 yards/min

Steps: 0

Calories: 614

Name: Run

Description: Running - 5 mph (12 min/mile)

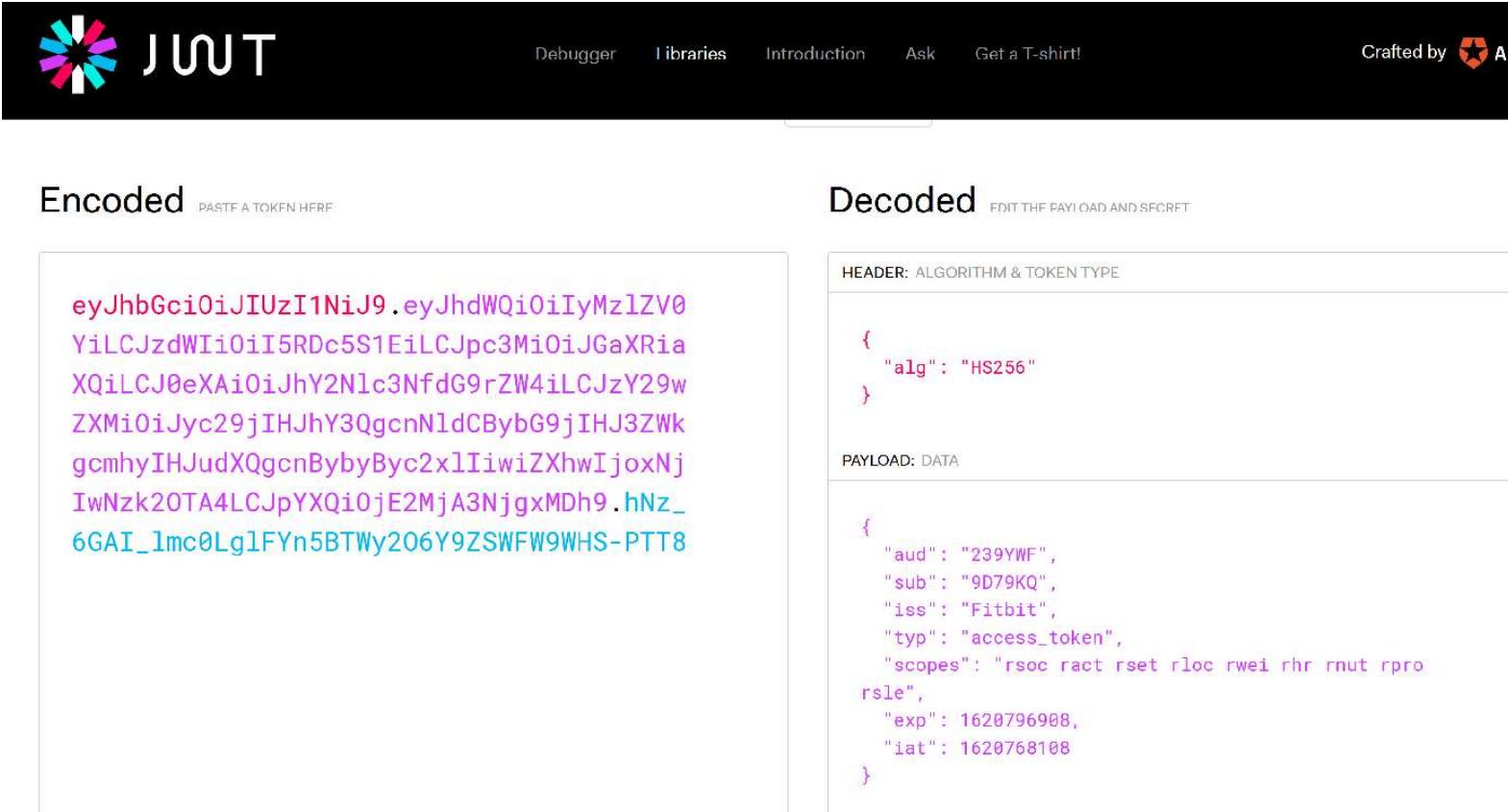
Steps: 6878

Calories: 478

Authorized

Cronologia attività						
Data	Attività	Passi	Distanza	Durata	Calorie	
Oggi, 18:00	Corsa	6.878	6,5 chilometri	1:10:40	478 cal.	Mostra dettagli
Oggi, 13:00	Nuoto	N/D	3000 meters	1:30:55	614 cal.	Mostra dettagli
Oggi, 07:00	Camminata	7.095	5 chilometri	1:00:50	170 cal.	Mostra dettagli

Il token



The screenshot shows a token decoding interface with two main sections: "Encoded" and "Decoded".

Encoded: PASTE A TOKEN HERE
eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiIyMzIzV0YiLCJzdWIiOiI5RDc5S1EiLCJpc3MiOiJGaXRiaXQiLCJ0eXAiOiJhY2Nlc3NfdG9rZW4iLCJzY29wZXMiOiJyc29jIHJhY3QgcnNldCBybG9jIHJ3ZWhgcmhyIHJudXQgcnBybyByc2xliwiZXhwIjoxNjIwNzk20TA4LCJpYXQiOjE2MjA3NjgxMDh9.hNz_6GAI_lmc0Lg1FYn5BTWy206Y9ZSWFW9WHS-PTT8

Decoded: EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "HS256"  
}
```

PAYOUT: DATA

```
{  
  "aud": "239YWF",  
  "sub": "9D79KQ",  
  "iss": "Fitbit",  
  "typ": "access_token",  
  "scopes": "rsoc ract rset rloc rwei rhr rnut rpro  
rsle",  
  "exp": 1620796908,  
  "iat": 1620768108  
}
```

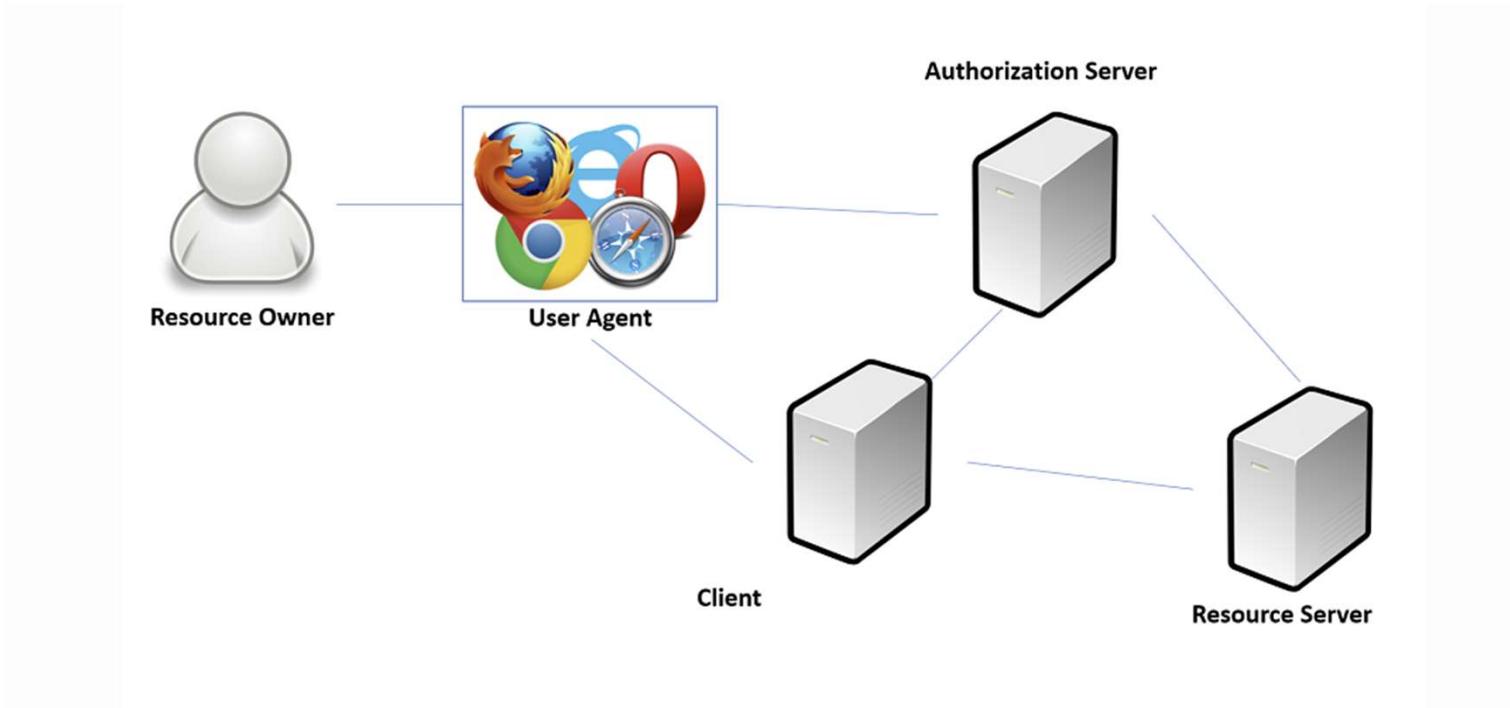
Autenticazione e autorizzazione con Oauth2 e il server Keycloak

Uso di JSON Web Token per gestire l'accesso alle risorse

Sommario

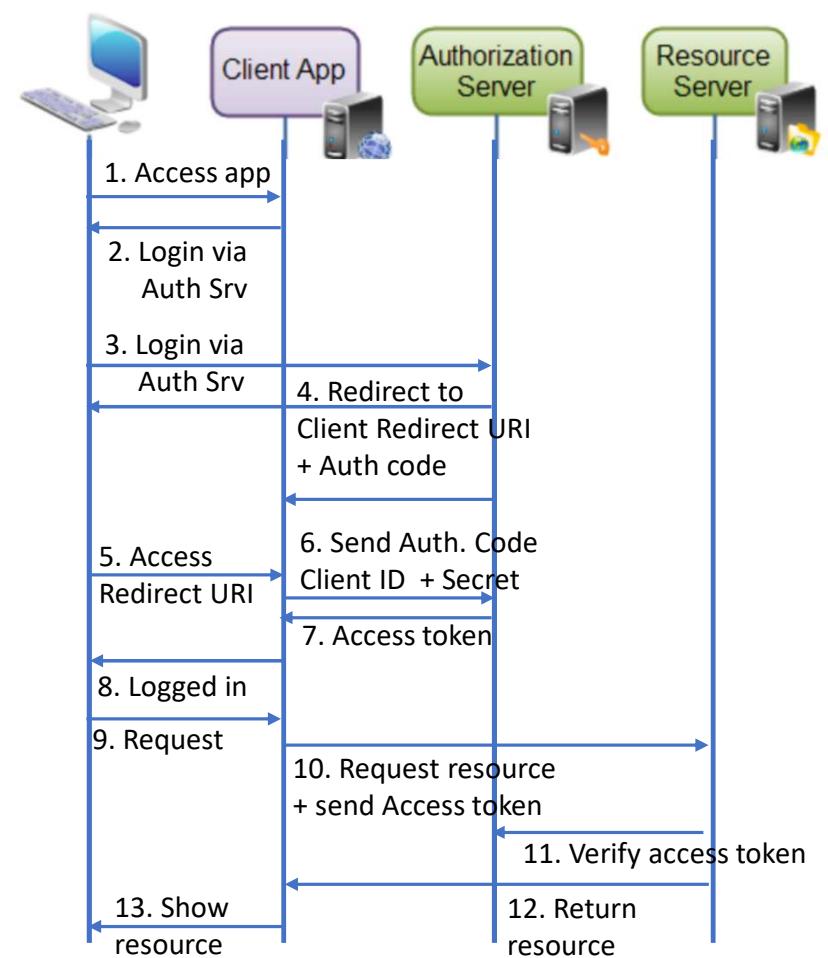
- Il protocollo OAUTH2
- JSON Web Token
- Il server Keycloak
- Applicazione al progetto

Richiamo sulla terminologia



Il protocollo

1. Il proprietario delle risorse (user) accede all'applicazione client.
2. L'applicazione client rinvia l'utente al login presso l'auth. server.
3. Per il login l'utente è ridiretto all'authorization server.
4. Quindi l'utente si autentica con l'authorization server. Se autenticato gli viene chiesto se intende concedere le risorse richieste all'applicazione. Se l'utente conferma è rinviato alla applicazione client (via Client Redirect URI) e con un Auth.Code
5. Il rinvio avviene mandando l'utente alla "redirect URI", specificato all'atto della registrazione. Insieme al rinvio, l'authorization server manda un codice che rappresenta l'autorizzazione (Auth. Code)
6. Quando si accede all'URI nell'applicazione client questa si connette direttamente all'authorization server. E invia il codice di autorizzazione, il client ID (ed eventualmente il segreto)
7. Se l'authorization server li ritiene validi ritorna un *token d'accesso*.
8. Lo user riceve conferma e può iniziare a fare le sue richieste alla client application (9)
10. L'applicazione client può ora usare il token per chiedere accesso alle risorse al resource server. Il token vale sia come autenticazione del client e del proprietario delle risorse (user), che come diritto di accesso alle risorse.
11. Il resource server può verificare la validità del token, in caso positivo risponde restituendo la risorsa (12, 13)



JSON Web Token (1/2)

- Il token usato per l'accesso può assumere diversi formati, uno dei più diffusi è il JWT – JSON Web Token (si pronuncia «jot»)
(per approfondimenti vedere jwt.io dove sono anche indicate varie librerie per la decodifica dei token, l'estrazione delle informazioni contenute nonché la verifica dell'autenticità del token)
- Il JWT comprende tre parti (codificate in BASE64 – base64url3):
 - Header
 - Payload (che potrà contenere varie informazioni, tra cui il tipo di permessi di accesso concessi con questo token)
 - Signature/Encryption data
- Il Resource Server può controllare l'autenticità del token verificando che il token sia firmato dall'Authorization Server (tramite la sua chiave pubblica).

JSON Web Token (2/2)

ESEMPIO:

```
eyJhbGciOiJIUzI1NilsInR5cCI6IkpXVCJ9.eyJzdWliOilxMjM0NTY3ODkwliwib  
mFtZSI6Ikpvag4gRG9IiwiWF0ljoxNTE2MjM5MDlyfQ.SflKxwRJSMeKKF2  
QT4fwpMeJf36POk6yJV_adQssw5c
```

DECODIFICATI:

Header: { "alg": "HS256", "typ": "JWT" }	Payload: { "sub": "1234567890", "name": "John Doe", "iat": 1516239022 }
---	---

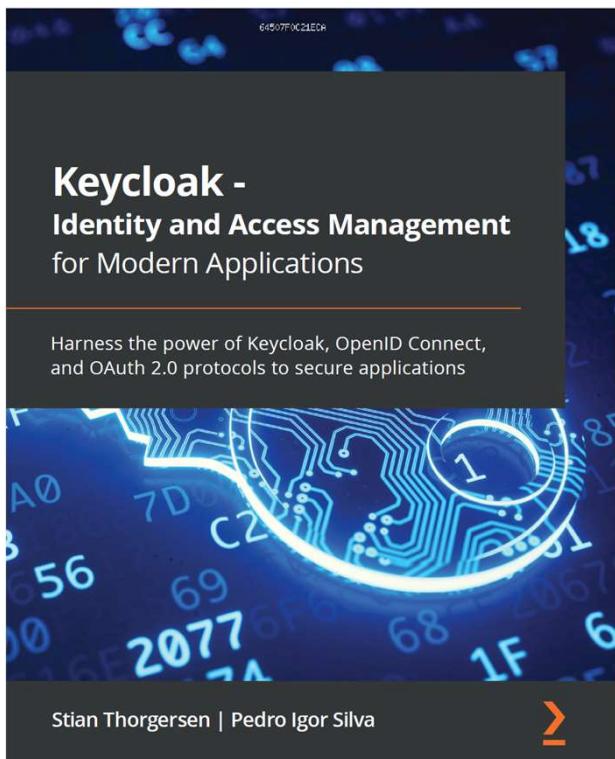
Collegandosi a jwt.io e inserendo il JWT sopra, potete verificarne l'autenticità inserendo la parola *secret* nel campo Verify Signature.

Un esempio di Authorization Server: Keycloak

- Keycloak è un authorization server che si può scaricare e sperimentare localmente
(per approfondire <https://www.keycloak.org/>)
- Potete attivare una istanza locale sulla vostra macchina ed utilizzare quella per testare il sistema

NOTE: Le ultimissime versioni hanno subito alcune modifiche, in particolare per quanto riguarda gli endpoint (le URL da utilizzare per accedere alle diverse funzioni del server) – quanto indicato di seguito è stato testato con keycloak versione 12 che è ancora disponibile per il download. I cambiamenti dovrebbero tuttavia riguardare solo le versioni più recenti, e gli esempi valere per versioni fino alla 17.

Un testo esplicativo con esempi



<https://github.com/PacktPublishing/Keycloak-Identity-and-Access-Management-for-Modern-Applications>

Keycloak – Identity and Access Management for modern Applications

Stian Thorgersen, Pedro Igor Silva

2021 Packt Publishing

Alcune funzionalità di Keycloak

- Permette di definire dei «realms» corrispondenti a contesti all'interno dei quali si possono registrare delle web-applications (la Client-App del diagramma di sequenza) che si appoggiano a keycloak (Authorization server nel diagramma di sequenza) per l'autenticazione (degli utenti) e per autorizzare l'accesso alle risorse (fornisce i token).
- Alle Client-App registrate si possono associare degli «scopes» che definiscono il tipo di permessi di accesso che saranno concessi dal Resource server (Es: potrebbe esserci uno scope di sola lettura, e uno che invece ha anche il permesso di modificare).
- All'interno di ogni *realm* si possono anche creare alcuni utenti ai quali possono essere associati ruoli per un controllo più fine degli accessi.

Installare keycloak

- Scaricare keycloak (per esempio la versione messa a disposizione sul dir) keycloak-12.0.4.zip
- Estrarre i file dallo zip
- Creare un utente amministratore (KC_HOME è il pathname della directory dove è stato estratto il file .zip)

To create an admin account on Linux or macOS, execute the following command in a terminal:

```
$ cd $KC_HOME  
$ bin/add-user-keycloak.sh -u admin -p admin
```

On Windows, execute the following command:

```
> cd %KC_HOME%  
> bin\add-user-keycloak.bat -u admin -p admin
```

Avviare keycloak su localhost

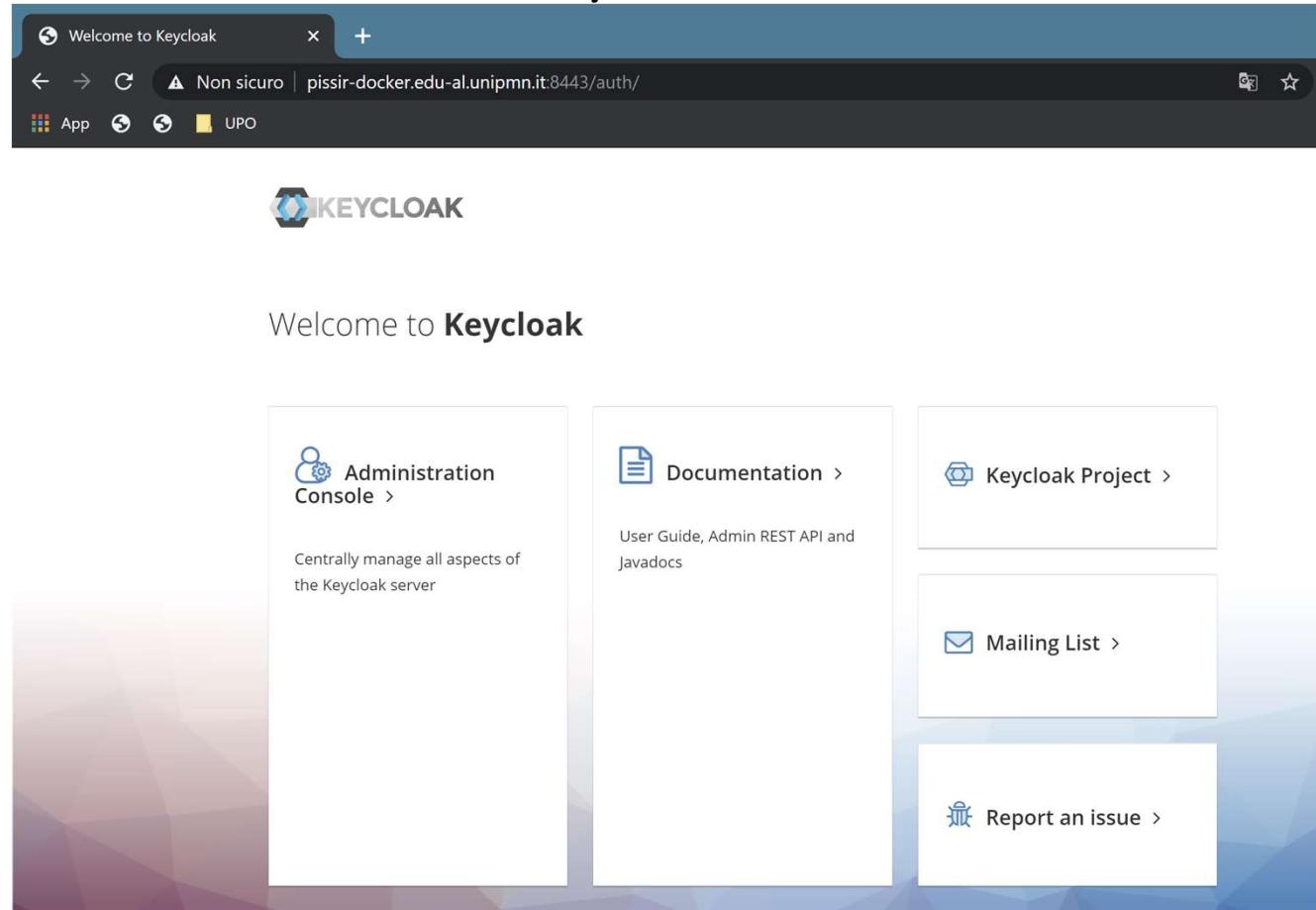
`.\bin\standalone.bat` oppure `./standalone.sh`

In questo modo il server viene avviato sulla porta 8080; se si vuole cambiare la porta su cui risponde si può aggiungere un parametro; per esempio per lanciare il server sulla porta 8443 ($8080 + 363 = 8443$)

`./standalone.sh -Djboss.socket.binding.port-offset=363`

Aprendo sul browser `localhost:8080/auth` si accede alla interfaccia di amministrazione

Interazione con Keycloak: la console di amministrazione



Quando installate keycloak per prima cosa occorre creare un utente admin che può **configurare i realms e registrare client applications ed utenti**.

I *realms*: contesti all'interno dei quali si attivano delle applicazioni (le client-app) e si registrano degli utenti

esempio di keycloak lanciato su localhost

The screenshot shows the Keycloak administration interface at `localhost:8443/auth/admin/master/console/#/realms/myrealm`. The left sidebar lists realms: 'Myrealm' (selected), 'MyDemo', and a button 'Add realm'. Other menu items include 'Client Scopes', 'Roles', 'Identity', 'Providers', 'User Federation', 'Authentication', and 'Manage'. The main panel is titled 'Myrealm' and contains tabs for 'General', 'Login', 'Keys', 'Email', 'Themes', 'Cache', 'Tokens', 'Client Registration', and 'Security Defenses'. The 'General' tab is active, showing fields for 'Name' (set to 'myrealm'), 'Display name' (set to 'MyRealm'), 'HTML Display name' (set to 'MyRealm'), 'Frontend URL' (empty), 'Enabled' (set to 'ON'), 'User-Managed Access' (set to 'OFF'), and an 'Endpoints' section with a link to 'OpenID Endpoint Configuration'.

The screenshot shows the Keycloak administration interface. On the left, a sidebar lists various management options like Configure, Realm Settings, Clients, Client Scopes, Roles, Identity, Providers, User Federation, Authentication, Groups, Users, Sessions, Events, and Import. The 'Clients' option is selected. The main area shows a breadcrumb path 'Clients > clientid-00'. The client details page for 'Clientid-00' is displayed, with the 'Settings' tab selected. The client configuration includes:

- Client ID: clientid-00
- Name: ClientTrial
- Description: Just to try it
- Enabled: ON
- Consent Required: OFF
- Login Theme: (empty dropdown)
- Client Protocol: openid-connect
- Access Type: confidential (selected in a dropdown menu)
- Standard Flow: (empty dropdown)
- Enabled: (checkbox checked)

A yellow box highlights the text "esempio di keycloak lanciato su localhost". A blue box highlights the "Access Type" dropdown, specifically the word "confidential".

Se il client è *confidential* c'è un «*secret*» (campo *Credentials*)

Le client-app (continua esempio)

Just to try it

Description ?

Enabled ?

Consent Required ?

Login Theme ?

Client Protocol ?

Access Type ?

Standard Flow Enabled ?

ON

Implicit Flow Enabled ?

OFF

Direct Access Grants Enabled ?

ON

Service Accounts Enabled ?

OFF

Authorization Enabled ?

OFF

Root URL ?

http://localhost:8080/

* Valid Redirect URLs ?

http://localhost:8080/*

Base URL ?

http://localhost:8080/*

+

Base URL ?

Qui si vede la URI dove viene
rinviato lo user una volta
completata la fase di
autenticazione

Gli utenti e i ruoli

KEYCLOAK

Myrealm

- Configure
 - Realm Settings
 - Clients
 - Client Scopes
- Roles**
- Identity
- Providers
- User Federation
- Authentication

Manage

- Groups
- Users
- Sessions

Roles

Realm Roles Default Roles

Role Name	Composite	Description	Actions	
new_role	False	Ruolo di prova	Edit	Delete
offline_access	False	\${role_offline-access}	Edit	Delete
uma_authorization	False	\${role_uma_authorization}	Edit	Delete

KEYCLOAK

Myrealm

- Configure
 - Realm Settings
 - Clients
 - Client Scopes
- Roles**
- Identity

Users

Lookup

ID	Username	Email	Last Name	First Name	Actions
14f4df9e-3da8-47...	myrealm_user2	giuliana.francesch...	Cognome	Nome	Edit
8924b047-a536-4...	myuser	giuliana.francesch...	User	My	Edit

Interrogare keycloak per ottenere JWT (confidential)

Vediamo alcune interazioni con keycloak via *curl* (caso di client-app *confidential*)

```
curl -X POST http://localhost:8080/auth/realms/H2Orealm/protocol/openid-connect/token -H 'content-type: application/x-www-form-urlencoded' -d 'username=clarabella&password=claraPWD2000&grant_type=password&client_id=ClientID-00&client_secret=4db97bce-6aec-4998-ba6a-f454b1c51560'
```

Si ottiene un JWT:

```
{"access_token":"eyJhbGciOiJSUzI1NilsInR5cClgOiAiSlDUliwia2IkliA6IC ... V56aOdZVh6gCrd60iuTA","expires_in":3600,"refresh_expires_in":1800,"refresh_token":"eyJhbGciOiJIUzI ... CSvRgPitiw","token_type":"bearer","not-before-policy":0,"session_state":"7e647a4c-346a-4ebe-88da-dddfa1bb29c5","scope":"email profile"}
```

(se anziché *confidential* la client app fosse *public* non si inserirebbe il *secret* nella richiesta)

Decifrare il JWT (su jwt.io)

Qui si possono vedere i contenuti decifrati dell'header e del payload.

Inserendo nel campo dedicato alla chiave pubblica ciò si ricava interrogando l'authorization server si verifica la signature.

<https://it.wikipedia.org/wiki/Base64>

```
{  
  "alg": "RS256",  
  "typ": "JWT",  
  "kid": "ZrgXzkehxCjleg770Es8xxGcYJ5RlqD1jICARH0dxv8"  
}  
  
PAYLOAD: DATA  
  
{  
  "exp": 1683878953,  
  "iat": 1683878653,  
  "jti": "691ef2b9-3ab5-4a2b-b356-f5cb32d9c73c",  
  "iss": "http://localhost:8080/auth/realm/H20realm",  
  "aud": "account",  
  "sub": "d47ff003-b79d-4611-a483-1765e4936279",  
  "typ": "Bearer",  
  "azp": "ClientID-00",  
  "session_state": "813b84cb-a16c-4f70-a5f0-2ffd195ebeac",  
  "acr": "1",  
  "allowed_origins": [  
    "http://localhost:8000"  
  ],  
  "realm_access": {  
    "roles": [  
      "offline_access",  
      "uma_authorization"  
    ]  
  },  
  "resource_access": {  
    "account": {  
      "roles": [  
        "manage-account",  
        "manage-account-links",  
        "view-profile"  
      ]  
    },  
    "scope": "email profile",  
    "email_verified": false,  
    "name": "Clarabella Giglioli",  
    "preferred_username": "clarabella",  
    "given_name": "Clarabella",  
    "family_name": "Giglioli",  
    "email": "giuliana.franceschinis@uniupo.it"  
  }  
}  
  
VERIFY SIGNATURE  
  
RSASHA256(
```



eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJacmdYWmtlaHhDamx1Zzc3T0VzOHh4R2NZsjVSbHFEMWpJQ0FSSDBkeHY4In0.eyJleHAiOjE2ODM4Nzg5NTMsImlhdCI6MTY4Mzg30DY1MywianRpIjoiNjkxZWYyjktM2FiNS00YTjlWIzNTYtZjVjYjMyZD1jNzNjIiwiXaNzIjoiaHR0cDovL2vxY2FsaG9zdDo4MDgwL2F1dGvcmVhbG1zL0gyT3J1YWxtIwiYXVkJoiYWNjb3VudCisInN1YiI6ImQ0N2ZmMDazLWI30WQtNDYxMS1hNDgzLTE3NjV1NDkzNjI3OSIsInR5cCI6IkJ1YXJlcIlsImF6cCI6IkNsawVuDE1ELTAwIiwc2Vzc21vb19zdGF0ZSI6IjgxM2I4NGNiLWEwNmMtNGY3MC1hNWyLTJmZmQxOTV1YmVhYyIsImFjciI6IjEiLCjhGxvd2VklW9yaWdpbnMiOlsiaHR0cDovL2vxY2FsaG9zdDo4MDAwI10sInJ1YWxtX2FjY2VzcyI6eyJyb2xlcYI6WyJvZmZsaW51X2FjY2VzcyIsInVtYV9hdXRob3JpemF0aW9uIl19LCJyZXNvdXJjZV9hY2N1c3Mi0nsiYWNjb3VudCI6eyJyb2xlcYI6WyJtYw5hZ2UtYWNjb3VudCisIm1hbmFnZs1hY2NvdW50LWxpbtmzIiwidm1ldy1wcm9maWx1Ii19fSwic2NvcGUiOj1bWFpbCBwcm9maWx1IiwiZW1haWxfdmVyaWZpZWQiOmZhbHN1LCJuYW11IjoiQ2xhcmFiZWxsYSBHaWdsaw9saSIsInByZWZlcnJlZF91c2VybmfTzSI6ImNsYXJhYmVsBEiLCJnaXZlbi9uYW11IjoiQ2xhcmFiZWxsYSIsImZhbw1seV9uYW11IjoiR2lnbG1vbGkiLCJ1bWFpbCI6ImdpdWxpYW5hLmZyYW5jZXNjaGluaXNAdW5pdXBvLml0In0.YoIRR_gKabpIi1qR0tYwWS8cIm_p62PNauqhWmWLhVtJG6E2JrwGF5Fd5847E-yi1-P6vdYNGNqqpSotg8aZmSYebIn1PzyW526ttvwhDVXR5JvTibf__kbtBZmPmBjcCqU02GDcdziPV-gFGa_wQZF_EQyhtKRCqn2tFaudWhwTs4ZLc8GfT411UmBI4vzFkzoYzQm7KeW0oU7_PhM9MDt15C41E21TTS3x4rYygeK8j-5FTCwNWdXc1XoaCbFaQ0uJ3zma3sMICiKdrDKfZKdDzwLIMZLuti1frZxvW7y2c_Wgs64diEPiZWg40nodG7jKgUZsqgauGIrpSdQ

Verificare il JWT: ottenere la chiave pubblica

```
curl http://localhost:8080/auth/realms/H2Orealm
```

```
{"realm":"H2Orealm","public_key":"MIIBIjANBgkqhkiG9w0BAQEFAOCAQ8AMIIBCgKCAQEAghTb8oN6ee5r/n0uccVVHfrfEaAAr34rGgMleBC3Y/nVezZv1VdQj6AWVuuGThGQF+YKit3A5+MT2qyhXQNZ+xv20ijeika6X2za7OFWMz7AlltNObZamiUQjg4mN6eOxgcw/vrtIQJTkaufsiun47Q2Dv4oxyEmd79N+BdRTm9693om0Ub7WZT2RryWPQO5drPEvwcMtqOs9CCy8bdaSpPTruxJRSq1UJs68f1XZVpKWGjkHSzc3bpmpGwVxJkIt58PZx9D20P64t4SnUFWi1mxBjxu5uqEKPS2wRB2OOQTxf4NSFZIMrYyGF43M64qnehJTHPOveE4bl6WxNjm8AwIDAQAB","token-service":"http://localhost:8080/auth/realms/H2Orealm/protocol/openid-connect","account-service":"http://localhost:8080/auth/realms/H2Orealm/account","tokens-not-before":0}
```

----- BEGIN PUBLIC KEY -----

```
MIIBIjANBgkqhkiG9w0BAQEFAOCAQ8AMIIBCgKCAQEAghTb8oN6ee5r/n0uccVVHfrfEaAAr34rGgMleBC3Y/nVezZv1VdQj6AWVuuGThGQF+YKit3A5+MT2qyhXQNZ+xv20ijeika6X2za7OFWMz7AlltNObZamiUQjg4mN6eOxgcw/vrtIQJTkaufsiun47Q2Dv4oxyEmd79N+BdRTm9693om0Ub7WZT2RryWPQO5drPEvwcMtqOs9CCy8bdaSpPTruxJRSq1UJs68f1XZVpKWGjkHSzc3bpmpGwVxJkIt58PZx9D20P64t4SnUFWi1mxBjxu5uqEKPS2wRB2OOQTxf4NSFZIMrYyGF43M64qnehJTHPOveE4bl6WxNjm8AwIDAQAB
```

----- END PUBLIC KEY -----



```
IjpmYWxzzSwibmFtZS16IK5vbWUgQ29nbm9tZS1s  
InByZWlcnJlZF91c2VybmfTzSI6Im15cmVhbG1f  
dXNlcjIiLCJnaXZlb19uYW1lIjoiTm9tZSIsImZh  
bWlseV9uYW1lIjoiQ29nbm9tZSIsImVtYWlsIjoi  
Z2l1bGlhbmcEuZnJhbmNlc2NoaW5pc0BkaS51bmlw  
bW4uaXQifQ.ZokRPlj1NuoyQZyU9ZOUfxj9A9C6m  
IZx8-W7s6fMqj5sB-u6Dg4fXbncjoIVvpp-  
CQR1E8DM9HbFMXppGaszcH8uMQkFGSwfmFsM-  
4jaRyr_iaSR2Yq-  
rPrKsN1Uu3QM9aLHBM46Q5YugvPQThYq1R_ipQMZ  
mC2Ln_JEZ541FcDquDX1sgfcHPuApG-Us14-  
KSDT5SA3cd8SsTNW1SiKxeJkXUnmpvAhKsjGkGOP  
EDODDDQXMD8LEHWFKecPtndhb19LIFgBS--  
u_BPnqDms5sBKnyMbT-  
N5ToivdbDLYmwUw7oJuyEcONBcLZQ0f4N7MMoV56  
a0dZVh6gCrd60iuTA
```

```
"resource_access": {  
    "account": {  
        "roles": [  
            "manage-account",  
            "manage-account-links",  
            "view-profile"  
        ]  
    }  
},  
"scope": "email profile",  
"email_verified": false,  
"name": "Nome Cognome",  
"preferred_username": "myrealm_user2",  
"given_name": "Nome",  
"family_name": "Cognome",  
"email": "giuliana.franceschinis@di.unipmn.it"
```

VERIFY SIGNATURE

```
RSASHA256(  
    base64UrlEncode(header) + "." +  
    base64UrlEncode(payload),
```

```
DJE0Rd9D1eD1LodseJ9oR0L02w  
NfxXz/dsw5e4zSJ7TEOV09C5o8J  
AjnJvEtpwIDAQAB  
-----END PUBLIC KEY-----
```

```
Private Key. Enter it in plain  
text only if you want to gener-  
ate a new token. The key never  
leaves your browser.
```

)

SHARE JWT

✓ Signature Verified

Verifica della signature

Inserita la chiave pubblica nel campo «public key» della sezione «VERIFY SIGNATURE» vediamo che è stato validato il JWT

Nota: la chiave dev'essere preceduta da

-----BEGIN PUBLIC KEY-----

e seguita da

-----END PUBLIC KEY-----

Osservazioni sul JSON contenente la chiave pubblica:

Sono indicati:

l'indirizzo dove richiedere il token (usato in precedenza in richiesta token):

"token-service":

`"http://localhost:8080/auth/realms/H2Orealm/protocol/openid-connect",`

l'indirizzo per l'autenticazione degli utenti registrati:

"account-service":

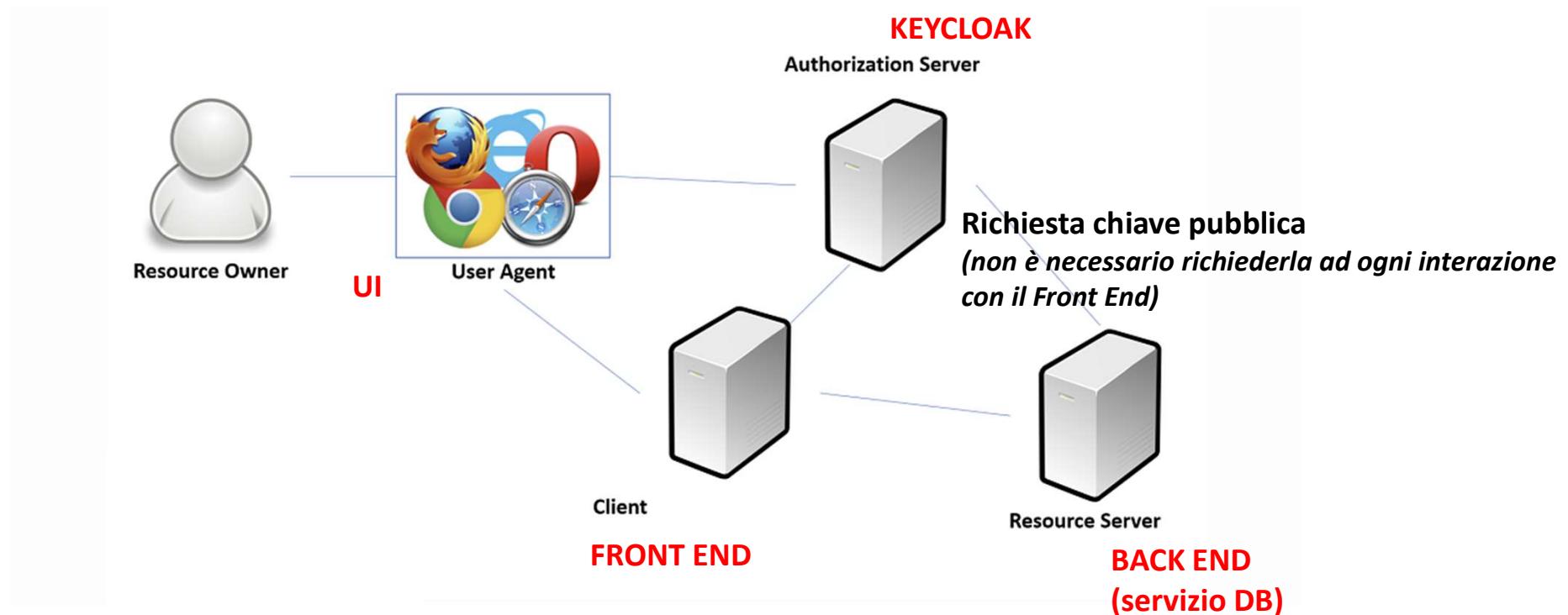
`"http://localhost:8080/auth/realms/H2Orealm/account"`

Autenticazione e autorizzazione con Oauth2, Open ID Connect tramite server Keycloak

Sommario

- Il protocollo OAUTH2
- JSON Web Token
- Il server Keycloak
- Applicazione al progetto
- Alcuni esperimenti

Come si applica al progetto



Esempio 1: applicazione NodeJS che usa Keycloak

Il primo esempio di applicazione NodeJS è descritta nel cap. 2 del libro – comprende un frontend ed un backend e mostra come si possano proteggere (alcune del)le funzionalità del backend limitando l'accesso ad utenti autenticati (tramite Keycloak)

Le due applicazioni si trovano nella cartella ch2 del repository scaricabile da github:

<https://github.com/PacktPublishing/Keycloak-Identity-and-Access-Management-for-Modern-Applications>

Su youtube trovate una playlist che mostra come avviare :

<https://www.youtube.com/playlist?list=PLeLcvrwLe187DykEKXg-9Urd1Z6MQT61d>

Una prima applicazione di esempio

Prima di iniziare: preparare un realm «myrealm» con un utente (scegliete voi un nome, nel mio esempio lo username è «keycloak») al quale è stato assegnato un ruolo «myrole» (creato per l'esempio all'interno di myrealm). Inoltre occorre registrare l'app «myclient» nel realm

The screenshot shows the Keycloak admin console interface. On the left, there's a dark sidebar with the title "Myrealm" and a dropdown arrow. Below it are several menu items: "Configure", "Realm Settings" (selected), "Clients" (highlighted with a blue bar), "Client Scopes", "Roles", "Identity Providers", and "User Federation". The main content area has a header "Clients > Add Client" and a sub-header "Add Client". It contains fields for "Import" (with a "Select file" button), "Client ID" (set to "myclient"), "Client Protocol" (set to "openid-connect"), and "Root URL" (set to "http://localhost:8000"). At the bottom are "Save" and "Cancel" buttons.

Figure 2.4 – Creating the client in the admin console

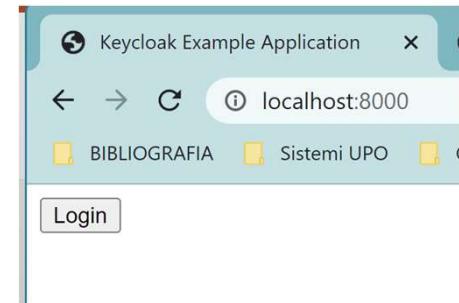
Una prima applicazione di esempio (avviare l'app)

- Avviare keycloak (sulla porta di default, la 8080)
- Eseguire all'interno delle due cartelle frontend e backend (occorre avere già installato nodeJS)
 - npm install
 - npm start
- Il frontend (la client app) si raggiunge tramite browser alla URL localhost:8000 mentre il backend è attivo sulla porta 3000

Sperimentare il primo esempio

FRONTEND

Su localhost:8000 si vede solo un bottone «login»



BACKEND

Su localhost:3000 ci sono 2 link:



- Se si tenta di fare login senza aver registrato l'app su keycloak, si otterrà un errore perché l'applicazione non viene riconosciuta.
- Cliccando su «Secured endpoint» lato backend si ottiene un errore «Access denied» - accedendo dal frontend previa autenticazione si riuscirà ad accedere alla parte protetta del backend

Autenticazione su keycloak

The screenshot shows a Firefox browser window. The address bar displays the URL: `localhost:8080/auth/realms/myrealm/protocol/openid-connect/auth?client_id=myclient&redirect_uri=http%3A%2F%2Flocalhost%3A8080%2Fmyapp%2F`. The main content area shows a "Sign in to your account" form with fields for "Username or email" and "Password", and a "Sign In" button. A white callout box with a black border and a diagonal line points from the text "Cliccando su Login si viene rediretti su Keycloak per l'autenticazione" to the "Sign In" button.

The developer tools Network tab is open, showing a list of requests. One request is highlighted in blue: a GET request to `localhost:8080/auth/realms/myrealm/protocol/openid-connect/auth?client_id=myclient&redirect_uri=http%3A%2F%2Flocalhost%3A8080%2Fmyapp%2F`. The status bar at the bottom of the browser indicates 16 requests, 1,31 MB transferred, and a total load time of 387 ms.

Protocollo	Metodo	Dominio	File	Iniziatore	Tipo	Trasferito	Dime...
00	GET	localhost:8080	auth?client_id=myclient&redi	keycloak.js:1345 (document)	html	5,28 kB	3,81 kB
00	GET	localhost:8080	all.css	stylesheet	x-unk...	NS_ERR...	0 B
00	GET	localhost:8080	base.css	stylesheet	css	6,12 kB	38,84 ...
00	GET	localhost:8080	app.css	stylesheet	css	52,31 kB	508,7...
00	GET	localhost:8080	patternfly.min.css	stylesheet	css	32,04 kB	182,7...
00	GET	localhost:8080	patternfly-additions.min.css	stylesheet	css	31,23 kB	225,0...

Filtre URL

Tutti HTML CSS JS XHR Caratteri Immagini Media WS Altro Disattiva cache Nessun limite

Header Cookie Richiesta Risposta Tempi Analisi dello stack

Filtre header Blocca Ritrasmetti

► GET http://localhost:8080/auth/realms/myrealm/protocol/openid-connect/auth?client_id=myclient&redirect_uri=http://localhost:8080/&state=c6505b65-919e-439b-87e5-e4d6616f4d7f&response_mode=fragment&response_type=code&scope=openid&nonce=7572510e-6a6e-49e6-8c32-b5b5dfd0a5a2

Stato 200 OK

Versione HTTP/1.1

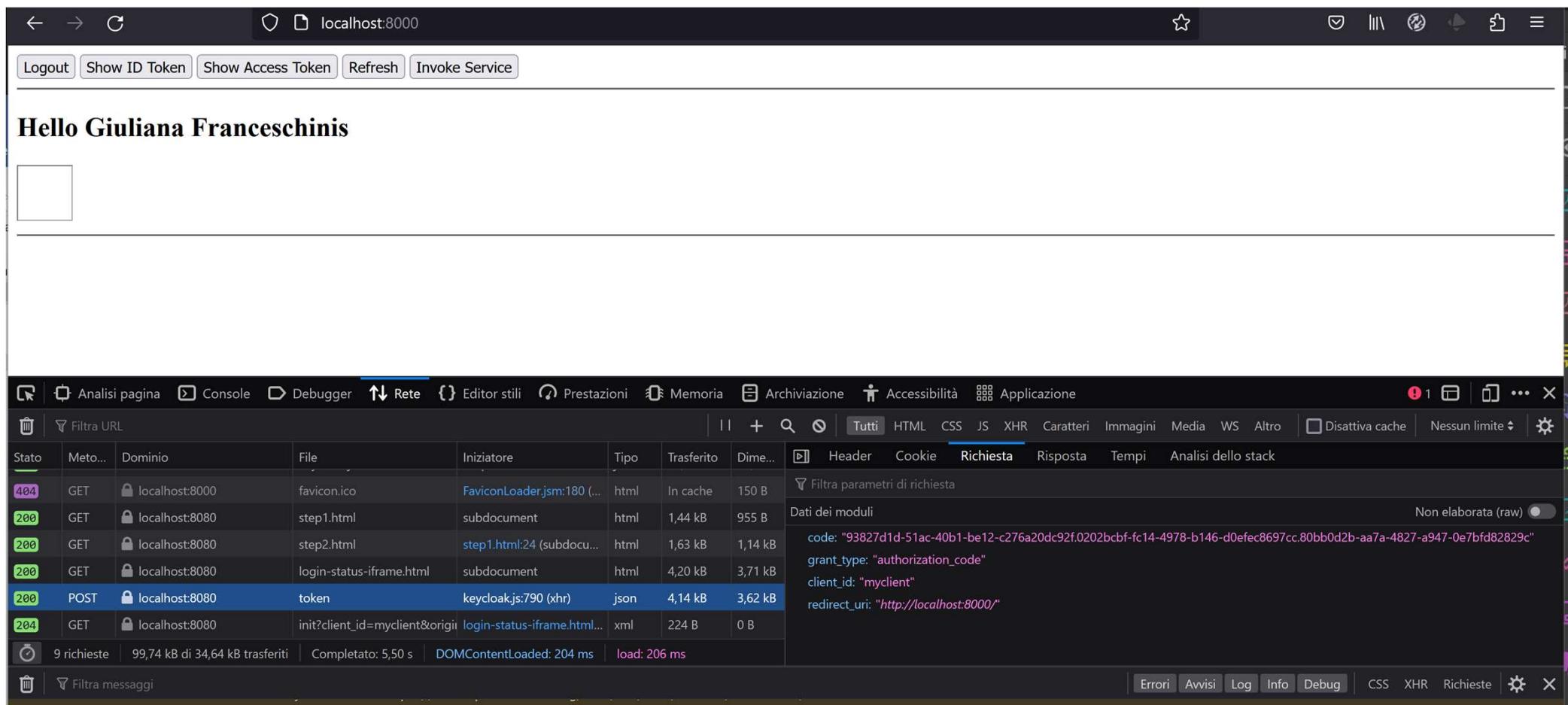
Trasferito 5,28 kB (dim. 3,81 kB)

Referrer Policy strict-origin-when-cross-origin

16 richieste 1,31 MB di 460,49 kB trasferiti Completato: 326 ms DOMContentLoaded: 58 ms load: 387 ms

Filtre messaggi Errori Avvisi Log Info Debug CSS XHR Richieste

Qui l'utente è autenticato, notiamo la richiesta del token



The screenshot shows a browser window with the University of Piemonte Orientale (UPo) logo at the top. Below it, a large heading reads "Qui l'utente è autenticato, notiamo la richiesta del token". The main content area displays a user profile with the name "Hello Giuliana Franceschinis" and a small placeholder image. At the bottom of the page, there is a navigation bar with links: Logout, Show ID Token, Show Access Token, Refresh, and Invoke Service.

The browser's address bar shows "localhost:8000". The developer tools Network tab is open, showing a list of requests. The last request in the list is a POST request to "localhost:8080/token" from "keycloak.js:790 (xhr)" with a "json" type, size "4,14 kB", and response size "3,62 kB". The response body contains the following JSON:

```
code: "93827d1d-51ac-40b1-be12-c276a20dc92f.0202bcf-fc14-4978-b146-d0efec8697cc.80bb0d2b-aa7a-4827-a947-0e7bfd82829c"
grant_type: "authorization_code"
client_id: "myclient"
redirect_uri: "http://localhost:8000/"
```

The Network tab also includes filters for Header, Cookie, Richiesta, Risposta, Tempi, and Analisi dello stack, and a search bar for "Filtro parametri di richiesta".



UNIVERSITÀ DEL PIEMONTE ORIENTALE

E l'access token in risposta da Keycloak

[Logout](#) [Show ID Token](#) [Show Access Token](#) [Refresh](#) [Invoke Service](#)

Hello Giuliana Franceschinis



Crea PDF

Rete Editor stili Prestazioni Memoria Archiviazione Accessibilità Applicazione

Analisi pagina Console Debugger Rete Editor stili Prestazioni Memoria Archiviazione Accessibilità Applicazione

Disattiva cache Nessun limite

Stato	Meto...	Dominio	File	Iniziatore	Tipo	Trasferito	Dime...
404	GET	localhost:8000	favicon.ico	FaviconLoader.jsm:180 (...	html	In cache	150 B
200	GET	localhost:8080	step1.html	subdocument	html	1,44 kB	955 B
200	GET	localhost:8080	step2.html	step1.html:24 (subdocu...	html	1,63 kB	1,14 kB
200	GET	localhost:8080	login-status-iframe.html	subdocument	html	4,20 kB	3,71 kB
200	POST	localhost:8080	token	keycloak.js:790 (xhr)	json	4,14 kB	3,62 kB
204	GET	localhost:8080	init?client_id=myclient&origi	login-status-iframe.html...	xml	224 B	0 B

Filtra proprietà

JSON

Non elaborata (raw)

```
access_token: "eyJhbGciOiJSUzI1NiIsInR5cClgOiAiSlidUiwia2IkIa6ICixLUQyMhhRUNpIMTzUTDdwd1hLWldGbj1OG4ySjVubG5MNmZGb1FWek7jh0eyJleHAIoJe2ODM4OTAzMDAslmIhdC16MTY4Mzg5MDAwMCwiYXVoF90aW1IjoxNjgzODkwMDAwLCjqdGkOjIY2IyNjM3NC0xMWZhlTQ2TUOGQ1NC1hZTU3ZmNjZGE2MWElCUpc3MiOjodHRwOi8vbG9jYWxob3N0OjgwODAvXXV0aC9yZWFSbXMvbXlyZWFSbSlsmF1ZCI6ImFjY291bnQiLCzdWliOi5Mjc3NzlyMC1kNzE5LTQzMWQtYTE2MS1mYWY3Mjg2MmMymElCJ0eXAiOjCZWfyzXliLCJhenAiOjteWNsaWVuWVudCislm5vbmlNljoInzU3MjUxMGUtNmE2ZS00OWU2LThjMzltyjViNRmZDBhNWEylwic2zc2l...mZXJyZWRfdXNlcm5hbWUiOjrzXljbG9haylsmdpdmVuX25hbWUiOjHaXvsWFuYSlsimZhbWiseV9uYW1IjoiRnjbhbmNlc2N0aW5ncvlsImVtYWlsInjz2I1hGlhbhbmEu7nJhbhbmNlc2NoaW5nc0RnbhWFnbhC5ib20ifOi87-K7AmHs3WM2r4alon4.fDWHn37lJTN
```

9 richieste 99,74 kB di 34,64 kB trasferiti | Completato: 5,50 s | DOMContentLoaded: 204 ms | load: 206 ms

Access token:

Nel payload del token vediamo il nome utente che viene visualizzato anche sul browser

HAi0jE20DM40TAzMAsIm1hdCI6MTY4Mzg5MDAw
MCwiYXV0aF90aW1lIjoxNjgzODkwMDAwLCJqdGk
i0iJlY2IyNjM3NC0xMWZhLTQ20TUtoGQ1NC1hZT
U3zmNjZGE2MWEiLCJpc3Mi0iJodHRwOi8vbG9jY
Wxb03N00jgwODAvYXV0aC9yZWFBsbXMvbXlyZWFS
bSIsImF1ZCI6ImfjY291bnQiLCJzdWIi0iI5Mjc
3NzIyMC1kNzE5LTQzMWQtYTE2MS1mYWY3Mjg2Mm
MyYmEiLCJ0eXAi0iJCZWfyzXIiLCJhenAi0iJte
WNsaVVudCIsIm5vbmn1joiNzU3MjUxMGUtNmE2
ZS00OWU2LThjMzItYjViNWVmZDBhNWeiyiwic2V
zc21vb19zdGF0ZSI6IjAyMDJiY2JmLWzJMTQtND
k30C1iMTQ2LWQwZwZ1Yzg20TdjYyIsImFjciI6I
jEiLCJhbGxvd2VkLW9yaWdpbnMiOlsiaHR0cDov
L2xvY2FsaG9zdDo4MDAwI10sInJ1YWxtX2FjY2V
zcyI6eyJyb2x1cI6WyJvZmZsaW51X2FjY2Vzcy
IsInVtYV9hdXR0b3JpemF0aW9uIiwbXlyb2x1I
119LCJyZXNvdXjJzV9hY2Nlc3MiOnsiYWNjb3Vu
dCI6eyJyb2x1cI6WyJtYW5hZ2UtYWNjb3VudCI
sIm1hbmFnZs1hY2NvdW50LWxpbtzIiwidmlldy
1wcm9maWx1I19fSwic2NvcGUi0iJvcGVuaWQgc
HJvZm1sZSB1bWFpbCIsImVtYWlsX3ZlcmlmaWV
IjpmYWxzZSwibmFtZSI6IkdpdWpxYW5hIEZyYW5
jZXNjaGluaXMiLCJwcVmZXJyZWRfdXN1cm5hbW
Ui0iJrZX1jbG9hayIsImdpdmVuX25hbWUi0iJHa
XVsawFuYSIsImZhbWlseV9uYW11IjoiRnhbmN1
c2NoaW5pcyIsImVtYWlsIjoiZ21bGlhbmEuZnJ
hbmN1c2NoaW5pc0BnbWFpbC5jb20ifQ.j8ZzKZA
mHs3WM2r4algp4_fDWHp3zUTN4Jbdx7r424Xz9Y
rFmpn4FEkFXqGf-
JUaUxamai9Exgcr9TDZAeA_ubsVwLjGp9mEWgX0
tF_-R8Tk17CYe-
82HVCdrFTEBNZbnJd97xjeMsgbdiH4RS82w1aAB
1p_4e0rn66sXQvXU0vLejUdTHr0cX4YEkGzJSX8
wOn0kIDjUyx4vHQFFXE8w0fkEgHqtmsNsFIIuLkq
v6+r1CgKm0K_nDTwR0

Not the token is intended for
"kid": "1-D20xRzH16nL7pwXKZWFn9u8n2J5nlnL6fFoQVzKc"
}

Payload: DATA

```
{  
  "exp": 1683890300,  
  "iat": 1683890000,  
  "auth_time": 1683890000,  
  "jti": "ecb26374-11fa-4695-8d54-ae57fccda61a",  
  "iss": "http://localhost:8080/auth/realm/myrealm",  
  "aud": "account",  
  "sub": "92777220-d719-431d-a161-faf72862c2ba",  
  "typ": "Bearer",  
  "azp": "myclient",  
  "nonce": "7572510e-6a6e-49e6-8c32-b5b5fd0a5a2",  
  "session_state": "0202bcf-fc14-4978-b146-  
d0efec0697cc",  
  "acr": "1",  
  "allowed_origins": [  
    "http://localhost:8000"  
  ],  
  "realm_access": {  
    "roles": [  
      "offline_access",  
      "uma_authorization",  
      "myrole"  
    ]  
  },  
  "resource_access": {  
    "account": {  
      "roles": [  
        "manage-account",  
        "manage-account-links",  
        "view-profile"  
      ]  
    }  
  },  
  "scope": "openid profile email",  
  "email_verified": false,  
  "name": "Giuliana Franceschinis",  
  "preferred_username": "keycloak",  
  "given_name": "Giuliana",  
  "family_name": "Franceschinis",  
  "email": "giuliana.franceschinis@gmail.com"  
}
```

VERIFY SIGNATURE

Hello Giuliana Franceschinis

Show Access Token

Il pannello mostrato dall'app permette di visualizzare il contenuto del token (che naturalmente è lo stesso ottenuto decodificando l'access token in jwt.io)

```
{  
  "exp": 1683890300,  
  "iat": 1683890000,  
  "auth_time": 1683890000,  
  "jti": "ecb26374-11fa-4695-8d54-ae57fccda61a",  
  "iss": "http://localhost:8080/auth/realm/myrealm",  
  "aud": "account",  
  "sub": "92777220-d719-431d-a161-faf72862c2ba",  
  "typ": "Bearer",  
  "azp": "myclient",  
  "nonce": "7572510e-6a6e-49e6-8c32-b5b5df0a5a2",  
  "session_state": "0202bcbf-fc14-4978-b146-d0efec8697cc",  
  "acr": "1",  
  "allowed_origins": [  
    "http://localhost:8000"  
,  
  "realm_access": {  
    "roles": [  
      "offline_access",  
      "uma_authorization",  
      "myrole"  
    ]  
  },  
  "resource_access": {  
    "account": {  
      "roles": [  
        "manage-account",  
        "manage-account-links",  
        "view-profile"  
      ]  
    }  
  }  
}
```

Invoke service: richiama la funzione protetta del backend

[Logout](#) [Show ID Token](#) [Show Access Token](#) [Refresh](#) [Invoke Service](#)

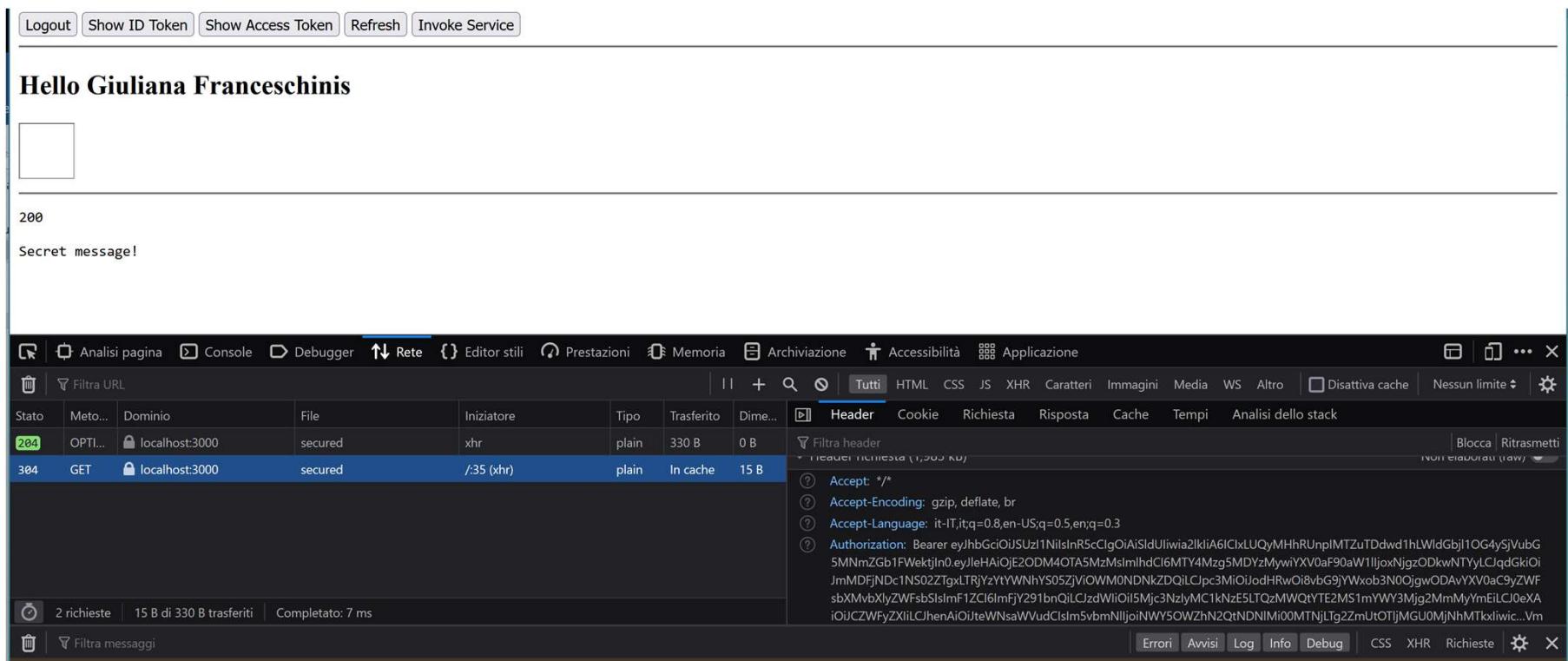
Hello Giuliana Franceschinis



200

Secret message!

Il token viene passato al backend nell'header della richiesta



The screenshot shows a web application interface and the browser's developer tools. The application displays a greeting and a placeholder for a secret message. The developer tools show a network request with the Authorization header containing a JWT token.

Application UI:

- Logout
- Show ID Token
- Show Access Token
- Refresh
- Invoke Service

Hello Giuliana Franceschinis

Secret message!

200

Developer Tools Network Tab:

Stato	Meto...	Dominio	File	Iniziatore	Tipo	Trasferito	Dime...
204	OPTI...	localhost:3000	secured	xhr	plain	330 B	0 B
304	GET	localhost:3000	secured	/:35 (xhr)	plain	In cache	15 B

Selected Request Headers:

- Accept: */*
- Accept-Encoding: gzip, deflate, br
- Accept-Language: it-IT, it;q=0.8, en-US;q=0.5, en;q=0.3
- Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cClgOiAiSlDUliwia2lkIA6lClxLUQyMHhRUlpIMTzUTDdwd1hLWldGbjI1OG4ySjVubG5MNrmZGb1FWektjln0eyJleHAQiE20DM40TA5MzMslmhdC16MTY4Mzg5MDYzMjwYXV0aF90aW1ljoxNjgzODkwNTYlCjqdGkiOijmMDfjNDc1NS02ZTgxLTRjYzYtYWNhYS05ZjViOWM0NDNkZDQiLCpc3MiOjodHRwOi8vbG9jYWxob3N0OjgwODAvYXV0aC9yZWFBsbXKvbXlyZWFsbslsmf1ZC16lmFjY291bnQlCzsdVliOit5Mjic3NzlyMC1kNzE5LTqzMWQyTE2MS1mYWY3Mjg2MmMyMymEilCj0eXAiOjCZWfVyzXzILChenaAiOjteWNsaWVudCIsIm5vbmNljojNWy50WZhN2QtNDNIMi00MTNjLtg2ZmUtOTijMGU0MjNhMTkxliwic...Vm

GLI STANDARD SU CUI SI BASA KEYCLOAK

Gli standard su cui si basa Keycloak (come molti altri sistemi di autorizzazione e autenticazione) sono:

- Oauth2.0 (per autorizzare un'applicazione ad accedere alle risorse di un dato utente)
- OpenID Connect (Per l'autenticazione degli utenti)
- JSON Web Tokens (JWT) – per la rappresentazione della *capability* di accesso alle risorse

AUTORIZZARE L'ACCESSO CON OAuth 2.0

Oauth2.0 è un framework che facilita l'integrazione di servizi web. Lo scopo è permettere l'accesso a risorse di utenti ad applicazioni di terze parti senza condividere con queste ultime le credenziali dell'utente, e consente anche un controllo fine su quali dati sull'utente vengono condivisi.

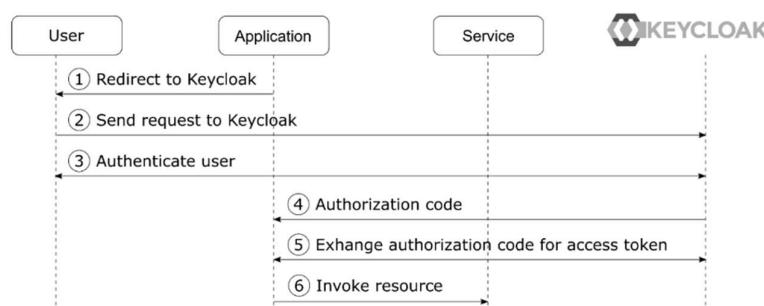
Per comprendere OAuth 2.0 Occorre definire i ruoli coinvolti nel protocollo di autorizzazione.

- **Il proprietario delle risorse:** tipicamente è l'utente finale, proprietario delle risorse (tipicamente dei dati) alle quali l'applicazione deve accedere;
- **Il Server dove si trovano le risorse:** questo è il servizio che gestisce le risorse protette;
- **L'applicazione Client:** questa è l'applicazione che deve accedere (in modo protetto ed eventualmente limitato) alle risorse;
- **L'Authorization Server:** questo è il server che emette i permessi (le capabilities) che autorizzano l'applicazione Client ad accedere alle risorse del proprietario delle risorse. Questo è il ruolo di Keycloak.

Il protocollo Oauth2.0 prevede i seguenti passaggi: l'applicazione Client chiede all'Authorization Server di accedere ad una determinata risorsa per conto del proprietario della stessa risorsa. L'Authorization Server emette un access token che consente l'accesso alla risorsa per un tempo limitato. Dopo aver ricevuto l'access token l'applicazione Client fa accesso al server dove si trovano le risorse presentando l'access token come prova dell'autorizzazione acquisita.

Ci sono diverse modalità per ottenere l'access token, di seguito sono indicate le possibili sequenze (flow) per l'acquisizione del token:

- Client credentials flow (si usa quando il client chiede l'autorizzazione ad accedere a proprie risorse)
- Device flow (si usa quando l'autorizzazione viene richiesta per conto di un dispositivo, cosa interessante per i sistemi IoT, lato sensori/attuatori sul campo, dove è impossibile prevedere l'inserimento di credenziali)
- Authorization Code flow: quest'ultimo è quello che sperimenteremo ed è illustrato attraverso un Message Sequence Chart nella figura seguente (tratta dal libro [1]):



Commentiamo brevemente i passi elencati nella figura precedente:

Figure 3.1 – OAuth 2.0 Authorization Code grant type simplified

1. La Client application prepara una richiesta di autorizzazione (il bottone di login sulla pagina iniziale dell'applicazione) e quando l'utente chiede di accedere, ridirige la sua richiesta sulla pagina per l'autenticazione di Keycloak.
2. Il browser dell'utente quindi si sposta su un endpoint del server Keycloak chiamato **authorization endpoint**
3. Se l'utente non si era già autenticato, allora Keycloak richiede all'utente di inserire le credenziali per consentire alla client application di fare accesso alle risorse protette a nome dell'utente
4. A questo punto la Client application riceve un authorization code da Keycloak all'interno di una authorization response
5. Utilizzando l'authorization code l'applicazione richiede all'Authorization Server di fornirgli un token di accesso, inviando una richiesta al **token endpoint** di Keycloak.
6. L'Authorization Server a questo punto invia il token alla client application che potrà finalmente accedere alle risorse secondo le richieste dell'utente, presentando il token ad ogni richiesta che rivolgerà al server che gestisce le risorse.

Il token ha una vita limitata, può essere fornito un refresh token per rinnovarlo senza dover chiedere nuovamente all'utente di riautenticarsi.

Ci sono alcuni ulteriori dettagli per garantire che solo determinate client application possano utilizzare questo meccanismo: le client applications devono essere registrate presso l'authorization server ed essere accessibili attraverso una URL predefinita (l'authorization server non accetterà una redirect URI diversa da quella registrata ed associata alla client application).

AUTENTICAZIONE DEGLI UTENTI CON OpenID Connect

La fase di autenticazione dell'utente avviene secondo lo standard Open ID Connect, un'estensione di Oauth2.0.

Questa funzionalità è quella che siamo abituati ad utilizzare quando usiamo per esempio le nostre credenziali Google o Facebook per accedere a servizi diversi e indipendenti da Google e Facebook. In questo modo gli utenti possono accedere a numerosi servizi senza dover gestire altrettante credenziali.

Anche OpenID Connect definisce alcuni ruoli:

- End User (l'equivalente del resource owner in Oauth2.0) – è la persona che deve autenticarsi;
- Relying party: si tratta dell'applicazione che vuole verificare l'identità dell'utente utilizzando un servizio di autenticazione esterno (quella che prima abbiamo chiamato client application)
- OpenID Provider: il provider che gestisce l'autenticazione dell'utente (e questo è il ruolo di Keycloak).

OpenID Connect usa il protocollo Authorization Code di OAuth2.0 ma quando avanza la richiesta specifica che ciò che viene richiesta è l'AUTENTICAZIONE e non l'AUTORIZZAZIONE ad accedere.

Tuttavia è possibile integrare le due cose e dopo l'avvenuta autenticazione ottenere anche un access token.

La sequenza è quella già vista in precedenza, e la client application può ricevere un ID token (in cambio dell'authorization code) e in questo modo dispone dell'identità dell'utente e può iniziare una sessione con l'utente dove quest'ultimo è autenticato.

Lo standard OpenID Connect definisce anche un preciso formato per l'ID token che può quindi essere letto direttamente dall'applicazione.

I JSON WEB TOKENS

L'uso di JWT ha diversi vantaggi: essendo in formato JSON è facile parsificarli, inoltre sono stati definiti diversi standard che permettono di scambiare token firmati per poter verificarne l'integrità. I token prevedono due parti "header" e "payload" (che contiene un insieme di "claims"), ed eventualmente una firma digitale. Le tre sezioni sono codificate in base64url-encoded e sono concatenate (con un "." di separazione tra la prima e la seconda, e tra la seconda e la terza).

Nel caso di token firmati, si potrà verificarne l'integrità utilizzando la chiave pubblica di chi ha emesso il token.

Per le operazioni di verifica dei token vale la pena di fare riferimento alle numerose librerie disponibili (es. vedere jwt.io per un elenco di librerie affidabili).

PASSI DI INSTALLAZIONE DI KEYCLOAK E ALCUNE OPERAZIONI DI AMMINISTRAZIONE

Per installare keycloak scaricare lo zip contenente il server (consiglio di utilizzare la versione messa a disposizione sul DIR: non è la più recente, ma è quella per cui il codice di esempio su cui faremo sperimentazioni è testato), estrarre il contenuto del file scaricato in una cartella, aggiungere un utente amministratore con il seguente comando (KC_HOME è il pathname della directory dove è stato estratto il file .zip):

To create an admin account on Linux or macOS, execute the following command in a terminal:

```
$ cd $KC_HOME  
$ bin/add-user-keycloak.sh -u admin -p admin
```

On Windows, execute the following command:

```
> cd %KC_HOME%  
> bin\add-user-keycloak.bat -u admin -p admin
```

e poi avviare il server con il seguente script:

.\bin\standalone.bat si Windows oppure ./standalone.sh su Linux o Mac

In questo modo il server risponde sulla porta 8080. Se si vuole cambiare porta usare la seguente opzione:

./standalone.sh -Djboss.socket.binding.port-offset=363 (in questo caso la porta sarà 8080+364 = 8443)

Ora collegarsi a: localhost:8080/auth per accedere agli strumenti di amministrazione.

DUE ESEMPI DI APPLICAZIONI CHE UTILIZZANO KEYCLOAK

Il testo [1] propone alcuni esempi di applicazioni che mostrano come utilizzare i servizi di keycloak, ed inoltre sono molto utili per capire i passaggi del protocollo descritti nelle precedenti sezioni.

Queste applicazioni di esempio si possono scaricare liberamente da github dal seguente link:

<https://github.com/PacktPublishing/Keycloak-Identity-and-Access-Management-for-Modern-Applications>

LA PRIMA APPLICAZIONE

La prima applicazione si può utilizzare come test per verificare il corretto funzionamento di Keycloak dopo una prima installazione. Dopo aver scaricato il codice da github, la prima applicazione si può trovare nella sottocartella ch2 (riferita al capitolo 2 del libro).

Il codice è stato testato e funziona con versioni di keycloak non aggiornatissime: nel frattempo sono stati cambiati alcuni dettagli ed in particolare alcuni endpoint di keycloak, per questo sul DIR potete trovare uno .zip contenente la versione 12.0.4 di keycloak alla quale il testo (e il codice di esempio) si riferiscono.

L'applicazione di esempio si compone di due parti: una client application (frontend) e un resource server (backend). Per poterle provare occorre aver installato keycloak e creato un “realm” myrealm, registrato una client application “myclient”, ed un utente “myuser” con uno specifico ruolo “myrole” (vedere le slide per alcune indicazioni su come effettuare queste operazioni: è anche possibile vedere le operazioni in alcuni video sul canale youtube indicato sul repository di github:

<https://www.youtube.com/playlist?list=PLLeLcvrwLe187DykEKXg-9Urd1Z6MQT61d>)

Il Frontend è una Single-Page Application implementata in JavaScript (usiamo Node.js) che prevede poche semplici azioni: login con Keycloak, mostra il nome utente estratto dal token, mostra l’ID token, mostra l’Access token, invoca un endpoint del backend che richiede il token per restituire una risorsa (fittizia). – risponde su localhost:8000 (e su keycloak l’applicazione sarà stata registrata con l’indicazione che la URI alla quale rinviare il controllo dopo l’autenticazione deve avere la forma http://localhost:8000/*)

Il backend è un server con interfaccia API REST, è anch’esso implementato in Node.js, e ha due soli endpoints: /public (non richiede autorizzazioni di accesso), /secured (richiede autorizzazioni di accesso e in particolare è accessibile solo ad utenti che abbiano ruolo “myrole” definito in “myrealm”). Per provare il suo funzionamento in modo diretto si può accedere a <http://localhost:3000> e si può subito verificare che il link verso il “public endpoint” risponde con un messaggio “Public message!” mentre il “secured endpoint” risponde con un messaggio di errore (Access denied!) perché la richiesta non è corredata di un access token valido (quello che invece il frontend fornirà previa autenticazione dell’utente).

Il frontend ha la possibilità di richiedere un nuovo access token, utilizzando il refresh token, per evitare di chiedere all’utente di riautenticarsi quando scade l’access token stesso. Insieme al nuovo access token si ottiene anche un nuovo refresh token e la procedura si può ripetere più volte.

E’ possibile osservare alcuni dei passaggi (in particolare lo scambio di authorization code e token tra client application e keycloak) osservando i messaggi scambiati tramite gli “strumenti per sviluppatori” del browser (sezione Rete).

Si può quindi provare ad esaminare il token estratto nel formato base64 su jwt.io per verificare la corrispondenza dei suoi contenuti con quelli mostrati dal frontend. E’ anche possibile verificare la validità del token prelevando la chiave pubblica di keycloak (riferita al realm “myrealm”) che si può ottenere tramite l’endpoint di keycloak: <http://localhost:8080/auth/realms/myrealm>

Per poter osservare da vicino tutti i passaggi è anche possibile sperimentare il programma che si può trovare nella sottocartella ch4 del software scaricato da github

BIBLIOGRAFIA:

[1] *Keycloak – Identity and Access Management for modern Applications*, Stian Thorgersen, Pedro Igor Silva, 2021 Packt Publishing

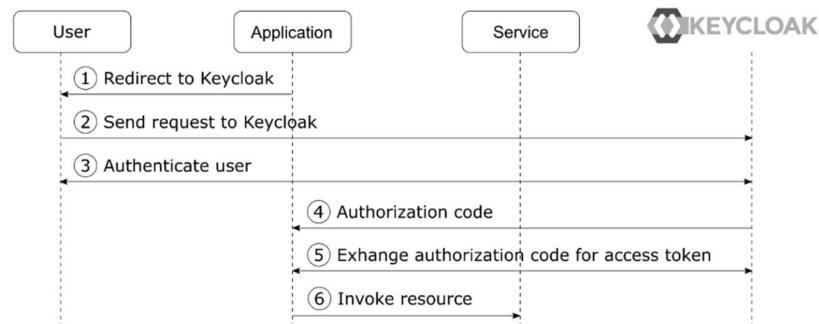


Figure 3.1 – OAuth 2.0 Authorization Code grant type simplified

ALCUNE PRECISAZIONI SUL PRIMO ESEMPIO: frontend-backend nella sottocartella ch2 del repository

<https://github.com/PacktPublishing/Keycloak-Identity-and-Access-Management-for-Modern-Applications>

Per questo esempio è necessario utilizzare il nome **myrealm** per il realm: infatti nella cartella ch2/backend si trova un file di configurazione keycloak.json dove è indicato "realm": "myrealm".

Inoltre per poter accedere da frontend all'esempio di servizio protetto del backend è necessario aggiungere al realm "myrealm" un ruolo "myrole" ed aggiungerlo all'utente registrato che si autenticherà. Se creare due user, uno con e uno senza questo ruolo nel primo caso dovrà ricevere una risposta positiva dal backend (cioè quando si preme sul bottone Invoke Service si ottiene la risposta "200 Secret message!"), nel secondo caso invece non sarà concessa l'autorizzazione ad accedere alla pagina protetta (cioè quando si preme sul bottone Invoke Service si ottiene la risposta "403 Access denied"). Per verificare la presenza del ruolo myrole, premendo su "Show Access Token" si può vedere il token e verificare che nella sezione "realm_access" – "roles" sia presente "myrole".

In questo stesso file è indicata la URL di keycloak: <http://localhost:8080/auth> pertanto l'applicazione si aspetta che keycloak sia fatto partire sulla porta di default, la 8080.

L'applicazione utilizza l'adapter Node.js messo a disposizione di keycloak la cui documentazione (per chi volesse approfondire) si trova qui: https://www.keycloak.org/docs/latest/securing_apps/#_nodejs_adapter

Per eseguire frontend e backend contenuti nella cartella ch2 (avendo già installato node js e il package manager npm):

- aprire un terminale nella cartella frontend
- eseguire npm install (questa operazione legge il contenuto di package.json e scarica in una sottocartella node_modules tutte le librerie necessarie)
- eseguire npm start (start è definito in package.json ed è semplicemente uno script che esegue "node app.js")
- ripetere i tre passaggi sopra per la cartella backend

Osserviamo i passi con il secondo programma di esempio

Nella cartella ch4 del repository scaricabile da

<https://github.com/PacktPublishing/Keycloak-Identity-and-Access-Management-for-Modern-Applications>

Troviamo un programma che può essere avviato con NodeJS:

npm install

npm start

Viene arrivato un servizio che risponde sulla porta 8000
(usare la URL localhost:8000 sul browser)



OpenID Connect Playground

1 - Discovery 2 - Authentication 3 - Token 4 - Refresh 5 - UserInfo Reset

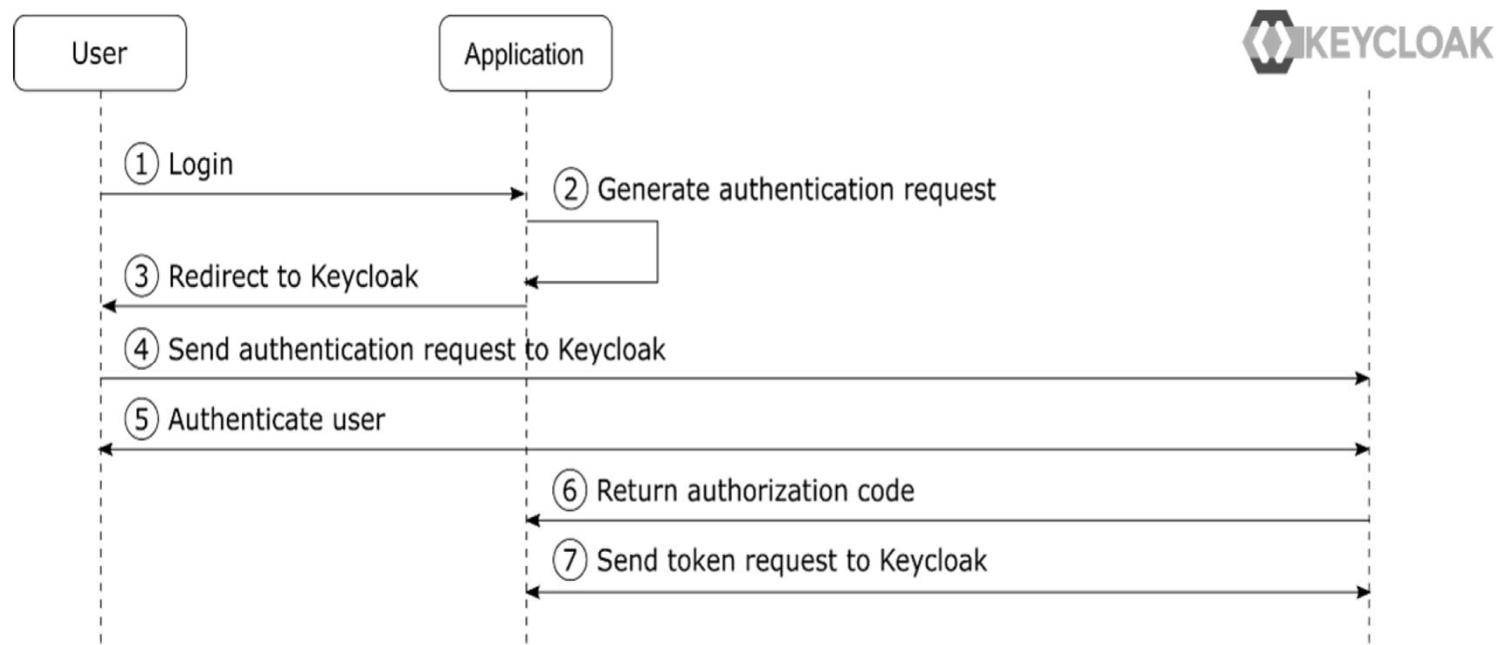


Figure 4.3 – The authorization code flow

Sequenza di passaggi che realizzano la sequenza del grant type **authorization code** e includono l'autenticazione su keycloak

1. Il primo passo permette di reperire gli endpoint da usare nei vari passaggi
2. Il secondo passo genera una richiesta di inizio della sequenza (che porterà all'autenticazione dell'utente su keycloak)
3. Il terzo passo permette di ottenere il token
4. Il quarto passo permette di «aggiornare» il token (refresh)
5. Nel quinto passo si vede come ottenere informazioni sull'utente tramite l'apposito endpoint

Interfaccia del programma di esempio



OpenID Connect Playground

[1 - Discovery](#) [2 - Authentication](#) [3 - Token](#) [4 - Refresh](#) [5 - UserInfo](#) [Reset](#)

Discovery

Issuer

[Load OpenID Provider Configuration](#)

OpenID Provider Configuration

Esito del primo passo: endpoint, grant type, algoritmi firma

Gli endpoint che verranno utilizzati nei passi successivi sono:

- "authorization_endpoint":

"`http://localhost:8080/auth/realm/myrealm/protocol/openid-connect/auth`",

- "token_endpoint":

"`http://localhost:8080/auth/realm/myrealm/protocol/openid-connect/token`",

"userinfo_endpoint":

"`http://localhost:8080/auth/realm/myrealm/protocol/openid-connect/userinfo`",

Grant type e response type

```
"grant_types_supported": [ "authorization_code", ... ]
```

- "response_types_supported": [
- "code", <<<<<< authorization code restituito
- ...
- "id_token", <<<<<< token contenente le informazioni sull'utente autenticato
- "token", <<<<<< access token – per poter accedere alle risorse dell'utente
- ...],

Sequenza di tipo «authorization code»

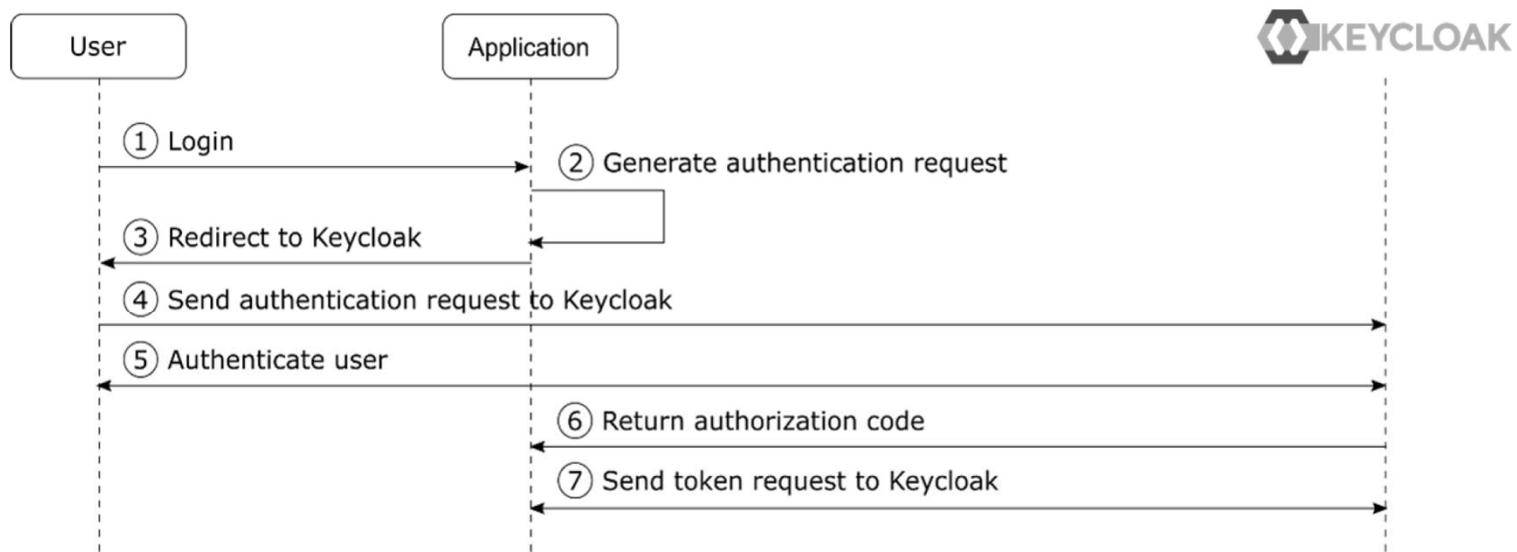


Figure 4.3 – The authorization code flow

OpenID Connect Playground

[1 - Discovery](#) [2 - Authentication](#) [3 - Token](#) [4 - Refresh](#) [5 - UserInfo](#) [Reset](#)

Authentication

client_id	<input type="text" value="oidc-playground"/>
scope	<input type="text" value="openid"/>
prompt	<input type="text"/>
max_age	<input type="text"/>
login_hint	<input type="text"/>

[Generate Authentication Request](#)

Authentication Request

[Send Authentication Request](#)

Authentication Response

Passi da 2 a 4 della sequenza

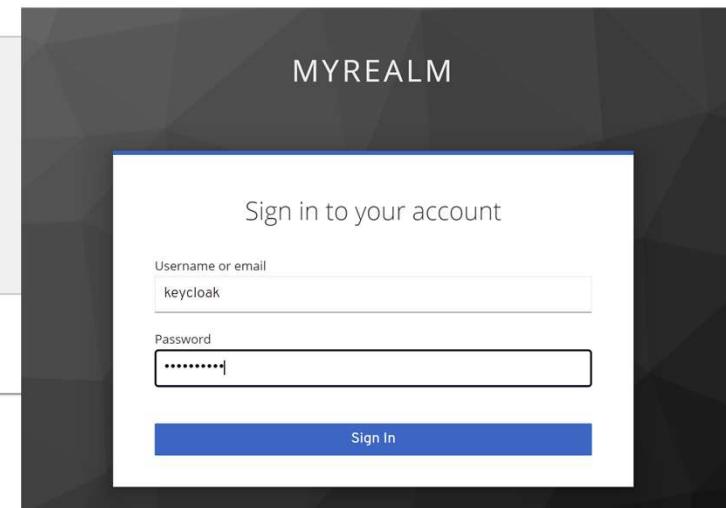
Generate Authentication Request – Send Authentication Request

Authentication Request

```
http://localhost:8080/auth/realms/myrealm/protocol/openid-connect/auth
```

```
client_id=oidc-playground
response_type=code
redirect_uri=http://localhost:8000/
scope=openid
```

[Send Authentication Request](#)



Authentication Response

```
code=bcbb470a-69d1-4b5d-ad5e-30bdfc7607e4.8f83dc7f-2008-4d81-a54d-604026383e63.2088eac8-450a-4071-a031-76bddf99acb6
```



OpenID Connect Playground

1 - Discovery 2 - Authentication 3 - Token 4 - Refresh 5 - UserInfo Reset

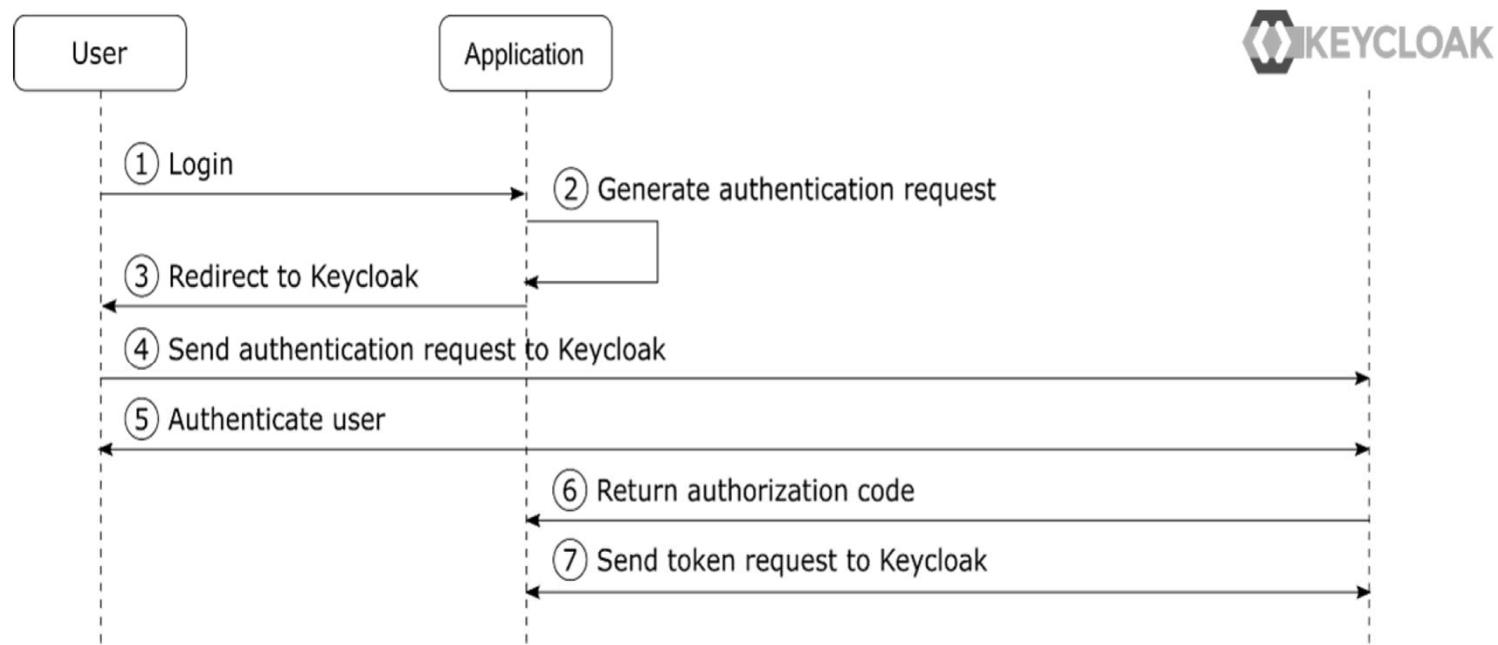


Figure 4.3 – The authorization code flow

PASSI 6-7 – richiesta del token utilizzando l'authorization code

Authorization Code

e216d3c9-cd32-419a-a026-73a9f8ad5767.f7c9874f-5829-470f-af12

Send Token Request**Auth. Code dal passo precedente**

Token Request

http://localhost:8080/auth/realms/myrealm/protocol/openid-connect/token
grant_type=authorization_code
code=e216d3c9-cd32-419a-a026-73a9f8ad5767.f7c9874f-5829-470f-af12-69a31d74a970.2088eac8-450a-4071-a031-76
client_id=oidc-playground
redirect_uri=http://localhost:8000/

Token endpoint

Token Response

```
{  
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICIxLUQyMHHRunpIMTzuTDwd1hLWldGbjl1OG  
  "expires_in": 300,  
  "refresh_expires_in": 1800,  
  "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJhYzI4OTI4OS01OGQzLTRmZDETYTQ5ZC1iZ  
  "token_type": "Bearer",  
  "id_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICIxLUQyMHHRunpIMTzuTDwd1hLWldGbjl1OG4ySj  
  "iss": "https://localhost:8080/auth/realms/myrealm"  
}
```

ID token
Access token
Refresh token

Scadenza del token:

```
{  
    "exp": 1684497331,  
    "iat": 1684497031,  
    "auth_time": 1684497024,  
    "jti": "83e7fdda-1235-444f-a778-df55b2d82897",  
    "iss": "http://localhost:8080/auth/realm/myrealm",  
    "aud": "oidc-playground",  
    "sub": "92777220-d719-431d-a161-faf72862c2ba",  
    "typ": "ID",  
    "azp": "oidc-playground",  
    "session_state": "f7c9874f-5829-470f-af12-69a31d74a970",  
    "at_hash": "WNzDwfmFH0E9J22EkP_uyg",  
    "acr": "1",  
    "yetanotherclaim": "My other claim",  
    "email_verified": false,  
    "realm_access": {  
        "roles": [  
            "offline_access",  
            "uma_authorization",  
            "myrole",  
            "newrole"  
        ]  
    },  
    "name": "Giuliana Annamaria Franceschinis",  
    "preferred_username": "keycloak",  
    "given_name": "Giuliana Annamaria",  
    "family_name": "Franceschinis",  
    "email": "giuliana.franceschinis@gmail.com",  
    "picture": "https://upobook.uniupo.it/Files/People/284/e51940e6-1732-4d2f-b0f9-cf07d25f5bae.png"  
}
```

<https://www.epochconverter.com/>

Convert epoch to human-readable date and vice versa

1684497331

Timestamp to Human date

[batch convert]

Supports Unix timestamps in seconds, milliseconds, microseconds and nanoseconds.

Assuming that this timestamp is in **seconds**:

GMT: Friday 19 May 2023 11:55:31

Your time zone: venerdì 19 maggio 2023 13:55:31 GMT+02:00 DST

Relative: 8 minutes ago

Refresh del token

Refresh

[Send Refresh Request](#)

Refresh Request

```
http://localhost:8080/auth/realms/myrealm/protocol/openid-connect/token  
grant_type=refresh_token  
refresh_token=eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJhYzI4OTI4OS01OGQzLTRmZDEtYTQ5ZC1iZTkwoT  
client_id=oidc-playground  
scope=openid
```

Refresh Response

```
{  
    "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICIxLUQyMhhRUNpIMTzuTDdwd1hLWldGbjl1O  
    "expires_in": 300,  
    "refresh_expires_in": 1800,  
    "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJhYzI4OTI4OS01OGQzLTRmZDEtYTQ5ZC1i  
    "token_type": "Bearer",  
    "id_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICIxLUQyMhhRUNpIMTzuTDdwd1hLWldGbjl10G4yS  
    "not-before-policy": 0,  
    "session_state": "f7c9874f-5829-470f-af12-69a31d74a970",  
    "scope": "openid profile email"  
}
```

Userinfo endpoint – richiedere dati utente

UserInfo Request

```
http://localhost:8080/auth/realms/myrealm/protocol/openid-connect/userinfo  
Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldU1...
```



Utilizzo il token

UserInfo Response

```
{  
    "sub": "92777220-d719-431d-a161-faf72862c2ba",  
    "email_verified": false,  
    "realm_access": {  
        "roles": [  
            "offline_access",  
            "uma_authorization",  
            "myrole",  
            "newrole"  
        ]  
    },  
    "name": "Giuliana Annamaria Franceschinis",  
    "preferred_username": "keycloak",  
    "given_name": "Giuliana Annamaria",  
    "family_name": "Franceschinis",  
    "email": "giuliana.franceschinis@gmail.com",  
    "picture": "https://upobook.uniupo.it/Files/People/284/e5...  
}
```

OPEN ID CONNECT

Grazie a questo standard (estensione di OAuth2.0 che mira a rendere disponibile servizi di autenticazione) i servizi su web possono delegare la gestione dell'autenticazione degli utenti ad un servizio esterno, con una semplificazione dei servizi che ne fanno uso, e riducendo il numero di credenziali che un utente deve gestire e il numero di volte che deve effettuare l'autenticazione.

Sono oramai molti i siti (e servizi) web che permettono di autenticarsi con l'account di google o dei social network. Vale la pena osservare che in ambito aziendale ciò consente il single-sign-on per tutti i propri sistemi.

UN SECONDO PROGRAMMA DI ESEMPIO PER ANALIZZARE IN DETTAGLIO TUTTI I PASSAGGI DEL PROTOCOLLO DI AUTENTICAZIONE VIA OpenID Connect e acquisizione del token per l'autorizzazione come previsto dallo standard OAuth2.0

Il programma è nella cartella ch4 del pacchetto scaricabile da github (associato al libro [1]):

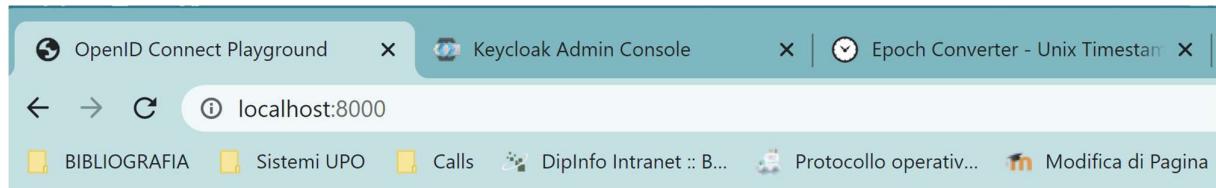
<https://github.com/PacktPublishing/Keycloak-Identity-and-Access-Management-for-Modern-Applications>

Il programma si lancia in Node JS spostandosi nella cartella ch4 e poi richiamando da linea di comando

```
npm install
```

```
npm start
```

Il servizio così attivato si raggiunge su <http://localhost:8000> e si presenta la seguente UI:



OpenID Connect Playground

[1 - Discovery](#) [2 - Authentication](#) [3 - Token](#) [4 - Refresh](#) [5 - UserInfo](#) [Reset](#)

Discovery

Issuer

[Load OpenID Provider Configuration](#)

OpenID Provider Configuration

Il primo passo “Discovery” permette di ricavare una serie di metadati che forniscono informazioni utili sugli endpoint di keycloak, sui tipi di protocolli autorizzativi (grant type) supportati e sugli algoritmi utilizzabili per le firme digitali che vengono apposte sui token.

L'informazione che si ottiene cliccando sul pulsante Load OpenID Provider Configuration è la stessa che si otterrebbe accedendo dal browser direttamente all'endpoint "Issuer" mostrato nella casella di testo: vediamone alcuni estratti:

```
"authorization_endpoint": "http://localhost:8080/auth/realm/myrealm/protocol/openid-connect/auth",
"token_endpoint": "http://localhost:8080/auth/realm/myrealm/protocol/openid-connect/token",
"introspection_endpoint": "http://localhost:8080/auth/realm/myrealm/protocol/openid-connect/token/introspect",
 userinfo_endpoint": "http://localhost:8080/auth/realm/myrealm/protocol/openid-connect/userinfo",
"end_session_endpoint": "http://localhost:8080/auth/realm/myrealm/protocol/openid-connect/logout",
```

Questi endpoint sono quelli da utilizzare per l'autenticazione e l'autorizzazione, per verificare il token, per ottenere informazioni sull'utente che si è autenticato, per chiudere una sessione (logout).

Per quanto riguarda i tipi di autorizzazione previsti, alcuni sono:

```
"grant_types_supported": [
  "authorization_code",    <<<<<< flusso di tipo authorization code
  ...
  "client_credentials" ],
"response_types_supported": [
  "code",      <<<<<< authorization code restituito
  ...
  "id_token", <<<<<< token contenente le informazioni sull'utente autenticato
  "token",     <<<<<< access token – per poter accedere a qualche risorsa dell'utente
  ...
],
```

Per quanto riguarda gli algoritmi per la firma digitale dei token, sono elencati i seguenti:

```
"token_endpoint_auth_signing_alg_values_supported": [
  "PS384", "ES384", "RS384", "HS256", "HS512", "ES256", "RS256", "HS384", "ES512", "PS256", "PS512", "RS512"
],
```

In aggiunta vengono elencati gli *scopes* supportati:

```
"scopes_supported": [
  "openid",
  "address",
  "email",
  "microprofile-jwt",
  "offline_access",
  "phone",
  "profile",
  "roles",
  "web-origins"
],
```

Ora sperimentiamo i vari passaggi della sequenza di tipo “authorization code”:

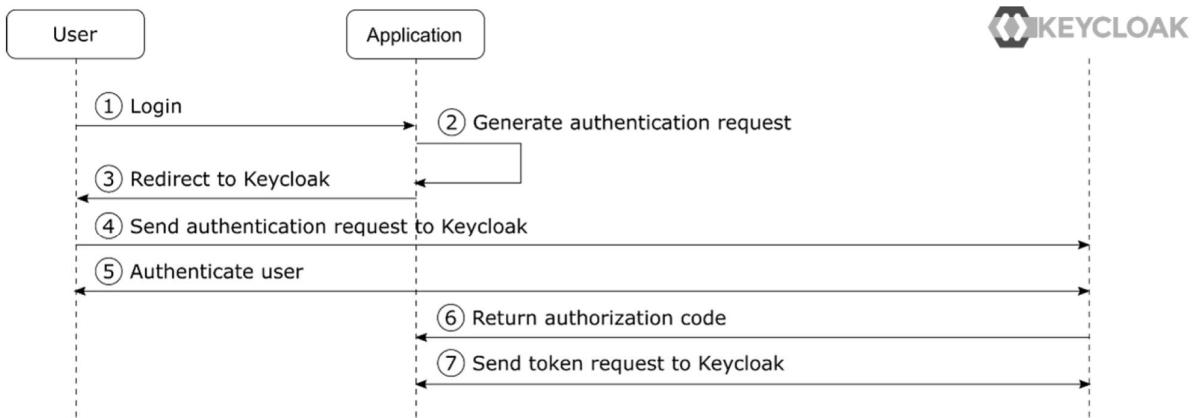


Figure 4.3 – The authorization code flow

Per generare la richiesta di autenticazione cliccare sul bottone 2-Authentication:

OpenID Connect Playground

1 - Discovery 2 - Authentication 3 - Token 4 - Refresh 5 - UserInfo Reset

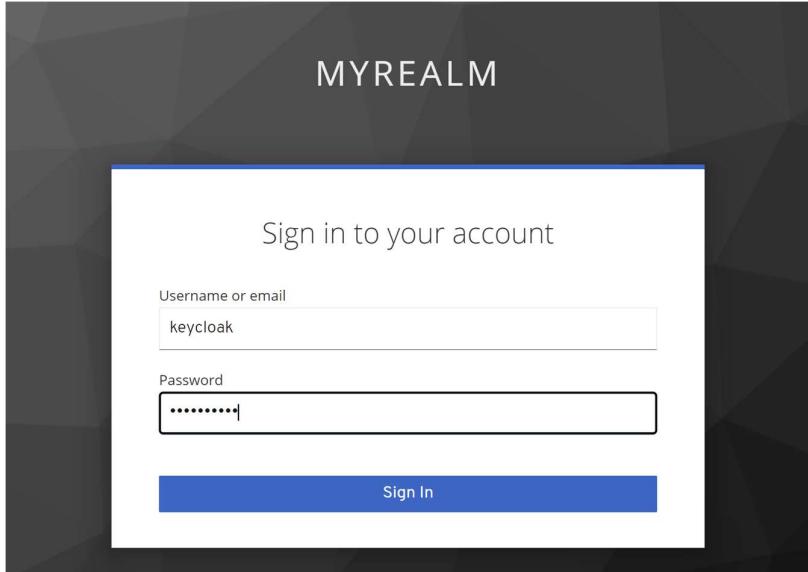
Authentication

client_id	<input type="text" value="oidc-playground"/>
scope	<input type="text" value="openid"/>
prompt	<input type="text"/>
max_age	<input type="text"/>
login_hint	<input type="text"/>

Authentication Request

Authentication Response

Inserendo il nome di una client-application (qui se ne sta usando una ad-hoc per questo esperimento, chiamata oidc-playground) e l'indicazione “scope id” = openid si può poi cliccare su “generate authentication request” per vedere cosa verrà inviato a kekcloak (passo 2 del diagramma di sequenza nella figura 4.3) e poi a seguire “send authentication request” (passo 4) che porta su Keycloak per l'inserimento delle credenziali:



Si possono osservare due cose: nell'authentication request si usa l'endpoint “authorization_endpoint” ricavato dai metadata, e nella richiesta si specificano il nome della client-application che effettua la richiesta, il tipo di risposta che ci si attende (code) e la URI a cui ridirigere il controllo una volta completato l'inserimento delle credenziali.

Authentication Request

```
http://localhost:8080/auth/realms/myrealm/protocol/openid-connect/auth  
client_id=oidc-playground  
response_type=code  
redirect_uri=http://localhost:8000/  
scope=openid
```

In risposta alla richiesta, dopo l'avvenuto inserimento delle credenziali si ottiene l'authorization code che ora la client application potrà reinviare a Keycloak per ottenere in cambio ID token ed eventualmente un access token (e refresh token associato)

Authentication Response

```
code=fc89ccc9-bcad-4bda-a691-4c67c7d4100e.6cd8c160-65c6-41c5-82b6-7a53b0c7a241.2088eac8-450a-4071-a031-76bddf99acb6
```

Passando ora alla sezione 3-Token request si può ottenere un token – attenzione perché l'authorization code ha una scadenza molto breve e se si lascia passare più di un minuto dalla authentication request si ottiene una risposta del seguente tipo:

Token Response

```
{  
  "error": "invalid_grant",  
  "error_description": "Code not valid"  
}
```

Se si invia l'authorization code rapidamente dopo averlo ricevuto, si ottiene il token: notiamo che si è usato il "token_endpoint", indicando come grant_type "authorization_code" più informazioni sull'id della client app che sta effettuando la richiesta.

Token Request

```
http://localhost:8080/auth/realms/myrealm/protocol/openid-connect/token  
grant_type=authorization_code  
code=563310e2-71f6-4934-beb1-538badc1256a.6cd8c160-65c6-41c5-82b6-7a53b0c7a241.2088eac8-450a-4071  
client_id=oidc-playground  
redirect_uri=http://localhost:8000/
```

Token Response

```
{  
    "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICIxLUQyMHhRUNpIMTzuTDwd1hLW]  
    "expires_in": 300,  
    "refresh_expires_in": 1800,  
    "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJhYzI4OTI4OS01OGQzLTRmZDET\  
    "token_type": "Bearer",  
    "id_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICIxLUQyMHhRUNpIMTzuTDwd1hLWldGb:  
    "not-before-policy": 0,  
    "session_state": "6cd8c160-65c6-41c5-82b6-7a53b0c7a241",  
    "scope": "openid profile email"  
}
```

La versione decodificata (base64-url) dell'ID token (che è un JWT) è mostrata sotto nella finestra: riconosciamo le tre sezioni del token: Header.Payload.Signature.

Nel Payload possiamo vedere varie informazioni tra le quali la scadenza ("exp") del token (che possiamo convertire in data e ora inserendolo per esempio in (<https://www.epochconverter.com/>)

Si vede a quale "realm" si fa riferimento (campo "iss") e il nome della client application che ha fatto la richiesta ("azp"), i ruoli che potranno essere usati per limitare eventualmente l'accesso da parte della client application, e poi un certo numero di dati anagrafici dell'utente che si è appena autenticato.

Nota: queste sono le stesse informazioni che si potrebbero ottenere inserendo la stringa dell'access_token nella finestra "Debug" del sito jwt.io

Per rinnovare un token scaduto, andando nella sezione 4-Refresh token, si può inviare una richiesta per ottenere un nuovo token (senza far ripetere l'inserimento delle credenziali all'utente). La richiesta differisce dalla precedente nell'authorization_grant che in questo caso è refresh_token, e nell'invio del refresh_token al posto del code.

Infine si può sperimentare l'uso dell'endpoint "userinfo" che restituisce le stesse informazioni sull'utente che sono anche contenute nell'ID token:

UserInfo Request

```
http://localhost:8080/auth/realms/myrealm/protocol/openid-connect/userinfo  
Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUI
```

UserInfo Response

```
{  
    "sub": "92777220-d719-431d-a161-faf72862c2ba",  
    "email_verified": false,  
    "realm_access": {  
        "roles": [  
            "offline_access",  
            "uma_authorization",  
            "myrole",  
            "newrole"  
        ]  
    },  
    "name": "Giuliana Annamaria Franceschinis",  
    "preferred_username": "keycloak",  
    "given_name": "Giuliana Annamaria",  
    "family_name": "Franceschinis",  
    "email": "giuliana.franceschinis@gmail.com",  
    "picture": "https://upobook.uniupo.it/Files/People/284/e5/  
}
```

Keycloak permette di aggiungere nuovi attributi e ruoli agli utenti, e nuovi scopes alla client application.

BIBLIOGRAFIA:

[1] *Keycloak – Identity and Access Management for modern Applications*, Stian Thorgersen, Pedro Igor Silva, 2021 Packt Publishing

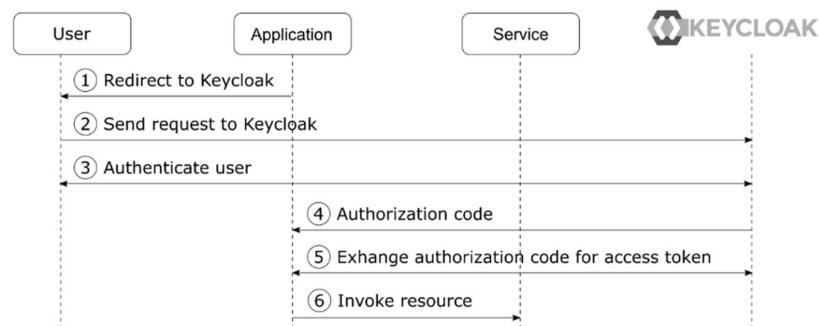


Figure 3.1 – OAuth 2.0 Authorization Code grant type simplified

Progetto d'esame: laboratorio di PISSIR

Anno accademico 2022/2023

DESCRIZIONE PROGETTO

Sistema di gestione di scorte d'acqua per l'irrigazione di coltivazioni: requisiti funzionali

Obiettivo del progetto è la progettazione e l'implementazione di un sistema di gestione di acqua per irrigazione di colture.

Gli utenti del sistema sono di due tipi: utenti autorizzati alla gestione delle colture di una specifica azienda, utenti che gestiscono le risorse idriche condivise dalle aziende.

I gestori delle risorse idriche possono aggiornare le informazioni relative alla disponibilità di risorse idriche e ai limiti massimi di erogazione giornaliera globali e per ciascuna azienda; inoltre possono consultare i dati relativi ai consumi (complessivi oppure dettagliati per azienda).

Ciascuna azienda possiede uno o più coltivazioni che devono essere irrigate: gli utenti autorizzati a gestire i dati di una data azienda possono inserire le informazioni relative alle coltivazioni possedute, che dovranno comprendere (almeno) la dimensione del campo (ettari), un parametro che indica se il tipo di coltivazione ha esigenze più o meno grandi di acqua (potrebbero essere per esempio 3 livelli: 1, 2, 3 per indicare poca, media, tanta), un parametro che indica il tipo di irrigazione (almeno 2, per indicare per esempio tradizionale o goccia a goccia ... potete indicarne altri) ed infine un parametro che indica il grado di umidità ideale da mantenere. In base a questi parametri e a misure di umidità e temperatura, ottenute periodicamente da sensori sul campo, si potrà ogni giorno stabilire la quantità d'acqua da erogare su ciascun campo. In funzione del numero e dell'estensione delle proprie coltivazioni, il gestore dell'azienda dovrà impostare le richieste di quantità giornaliera di acqua da utilizzare per l'irrigazione. Tale quantità può essere modificata periodicamente: in caso di aggiornamento la nuova quantità potrà essere accettata o rifiutata dal sistema di gestione delle risorse idriche in base alla disponibilità complessiva d'acqua ed alle richieste complessivamente avanzate da tutte le aziende; la nuova quantità, se accordata, sarà disponibile a partire dal giorno successivo alla conferma di variazione.

L'acqua prenotata viene pagata in ogni caso, anche se non utilizzata (per cui è interesse di ogni azienda fare una stima piuttosto precisa delle esigenze, ed è inoltre utile poter mantenere e consultare uno storico delle quantità prenotate ed effettivamente utilizzate per ciascuna coltivazione nel passato, oltre allo storico delle misure di umidità e temperatura rilevate nel tempo).

Oltre ai dati sulle coltivazioni si dovranno registrare le informazioni sui sensori e attuatori presenti nelle varie coltivazioni, in modo da poter associare le misure ricevute alla coltivazione dove si trova il sensore, e in modo da poter attivare (automaticamente) gli attuatori (irrigatori) nel momento in cui le condizioni (di temperatura ed umidità) lo richiedono.

Struttura del sistema (progettazione)

Il sistema dovrà essere composto da

- un "backend" che espone un'interfaccia di tipo REST e permette di inserire in un database i dati rilevanti sulle risorse idriche disponibili e la loro assegnazione alle aziende, e sulle informazioni relative alle aziende ed alle loro coltivazioni (incluse quelle che variano dinamicamente, come per

esempio i valori delle misure rilevate), ed inoltre permette di modificare o consultare tali dati. Il backend ha accesso esclusivo al DB (chiunque voglia aggiornare o leggere i dati deve chiederlo al backend).

- Una interfaccia utente (un'app, una web-app o una mobile app) che permetterà agli utenti di interagire con il backend (tramite le API-REST di quest'ultimo). Questa potrà essere una semplice interfaccia testuale oppure una interfaccia a finestre (per esempio eseguibile nel browser, quindi una web-app) o una mobile app (se avete seguito il corso di applicazioni mobili).
- Un gestore del sottosistema IoT: si tratta di un componente che può entrare in comunicazione con i sensori e gli attuatori, invia periodicamente le misure rilevate tramite i sensori e agisce sullo stato degli attuatori in base ai requisiti di umidità stabiliti, alle misure rilevate e alla quantità d'acqua effettivamente disponibile. L'idea è che ogni azienda agricola di norma abbia nella propria rete locale (la stessa alla quale si agganciano i dispositivi sensori/attuatori) un gestore di sottosistema IoT. I gestori dei sottosistemi IoT dovranno interagire con gli altri sottosistemi tramite broker MQTT (per esempio per trasmettere misure o indicazioni di quantità d'acqua consumate nell'irrigazione, per ricevere indicazione della quantità d'acqua effettivamente disponibile per la specifica coltivazione, e per inviare eventuali notifiche di allarme per insufficiente disponibilità d'acqua oppure, al contrario, per indicare un surplus/avanzo rispetto alla richiesta iniziale).

COMPONENTI ACCESSORI (facoltativi):

Il sistema potrebbe inglobare anche alcuni componenti accessori: seguono alcuni esempi.

- un sistema esterno di autenticazione / autorizzazione (es. keycloak) da utilizzare per permettere l'accesso ai soli utenti autorizzati (previa autenticazione);
- un sistema esterno che fornisce le previsioni meteo in modo da poter basare le richieste di acqua giornaliera anche in funzione della probabilità di avere pioggia, oppure che le temperature siano molto alte;
- un "mercato dell'acqua" che potrebbe permettere ad aziende che hanno un esubero di acqua di cedere "crediti d'acqua" a chi invece non riesce a coprire tutte le proprie esigenze di irrigazione.

SENSORI E ATTUATORI: i sensori e gli attuatori potranno essere reali o emulati, in base alla disponibilità di strumenti fisici. In ogni caso per poter effettuare più facilmente i test sul sottosistema IoT e i test d'integrazione occorrerà predisporre delle piccole applicazioni in grado di fornire misure fittizie (che siano rappresentative di scenari realistici, e che potrebbero essere memorizzate e quindi lette da un file: ciò sarebbe molto utile sia per lo svolgimento dei test che in occasione della demo da mostrare all'esame) ma anche di recepire i comandi per gli attuatori e visualizzarli in formato testo o grafico (una possibilità è usare l'emulatore delle Philips Hue assegnando a ciascuna lampadina emulata un significato, come rappresentante di un attuatore fisico, e in base al comando ricevuto accenderla, spegnerla o farle cambiare colore).

FASI DEL LAVORO

- **Specifiche:** in base ai requisiti elencati sopra, eventualmente dettagliati grazie a interviste che i gruppi potranno organizzare con il committente (i docenti del corso), occorrerà definire con sufficiente dettaglio il dominio e i casi d'uso. Per evitare ambiguità si dovranno usare schemi chiari preferibilmente utilizzando i diagrammi UML (diagramma delle classi del dominio e diagramma dei casi d'uso, corredati di brevi commenti testuali complementari ai diagrammi che spiegano in cosa consiste il caso d'uso; a vostra discrezione potrete usare altri tipi di diagrammi per specificare meglio l'articolazione in termini di successione di passi dei diversi casi d'uso).
- **Progettazione:** in questa fase si dovrà scegliere il tipo di architettura (che dovrà essere basata su microservizi), si dovranno individuare i sottosistemi, le loro interfacce e le interazioni. Inoltre

all'interno di ogni gruppo si dovranno definire dei sottogruppi a cui attribuire la responsabilità della progettazione di 1 o 2 sottosistemi (AI GRUPPI PIU' NUMEROSI SARA' RICHIESTO DI REALIZZARE UNA DELLE FUNZIONALITA' ACCESSORIE). Per ciascun sottosistema dovrà essere preparato un documento che definisca in modo sufficientemente preciso le classi che si dovranno implementare nella fase successiva: per evitare ambiguità la documentazione dovrà contenere schemi chiari, possibilmente utilizzando diagrammi UML. In particolare dovete schematizzare il diagramma delle classi (questa volta sarà il diagramma delle classi che dovranno essere implementate) e dei package, e i diagrammi di sequenza per descrivere le interazioni tra i componenti (sia all'interno di uno stesso microservizio, sia tra microservizi diversi). In questa fase si dovranno anche definire gli end-point delle API REST, con gli eventuali parametri e il formato delle informazioni scambiate, sia i topic MQTT con il formato dei messaggi inviati sui diversi topic

- **Implementazione:** Consigliamo di implementare in Java le classi definite in fase di progettazione, ma potete usare altri linguaggi a voi più congeniali (il supporto da parte dei docenti potrebbe non essere ugualmente proficuo se si scelgono altri linguaggi). I diversi microservizi possono essere implementati con linguaggi diversi (ciascun sottogruppo può implementare nel linguaggio preferito il proprio microservizio: l'importante è che le interfacce siano rispettate).
- **Test:** Lo sviluppo dovrà prevedere anche il test dei componenti e dei sottosistemi in isolamento. Quando il gruppo avrà completato tutti i sottosistemi dovrà svolgere anche test di integrazione. I test dovranno coprire un numero adeguato di casistiche tipiche, e saranno anche la base per la dimostrazione di funzionamento che dovrà essere illustrata durante la parte dell'esame relativa al progetto.
- **Demo:** durante l'esame sulla parte pratica dovrà essere preparata una demo che permetta di mostrare tutte le funzionalità implementate e che sia esempio significativo di alcuni dei test più rilevanti realizzati prima della consegna.

Specifiche: classi dominio e use case

Per disegnare i diagrammi UML per Use Case e Classi del dominio potete utilizzare Visual Paradigm

UTENTI:

- Gestori servizio idrico (GI)
- Gestori singole aziende (GA)

COLTIVAZIONI

- Dimensione
- Esigenza/richiesta acqua
- Grado umidità da mantenere

GA: Inserisce/visualizza coltivazione, inserisce/visualizza sensore, inserisce/visualizza attuatore, imposta richiesta d'acqua giornaliera, invia a GI richiesta acqua per il giorno successivo

Il GA monitora le misure rilevate dai sensori per la coltivazione (report su dati storici o in tempo reale) e monitora lo stato degli irrigatori

SENSORI (umidità, temperatura) periodicamente inviano MISURE

ATTUATORI, COMANDI (accensione/spegnimento irrigazione)

ATTUATORI ricevono comandi e CAMBIANO STATO di conseguenza

Il GI riceve, registra e visualizza (acqua totale disponibile,) le richieste giornaliere delle diverse aziende con la possibilità di accordarle/rifiutarle

Attore: Gestore Azienda (GA)

Caso d'uso: Gestione coltivazioni

1. Il GA chiede di visualizzare l'elenco delle coltivazioni della sua azienda agricola
2. Il GA sceglie l'azione da compiere: nuovo inserimento, modifica, cancellazione
(nel secondo e terzo caso deve anche selezionare una delle coltivazioni).

Estensioni:

- 2.a nuovo inserimento: il GA definisce i dati della coltivazione (dimensioni, esigenze d'acqua, umidità da mantenere) Al termine conferma e i dati della nuova coltivazione vengono registrati.
- 2.b modifica: al GA viene mostrata una scheda con i dati della coltivazione e può modificare i vari campi. Al termine conferma le modifiche e i nuovi dati vengono registrati.
- 2.c eliminazione: viene chiesta conferma dell'eliminazione e in caso affermativo i dati vengono cancellati.

Attore: Sensore

Caso d'uso: Invio misure

1. Il sensore rileva continuamente le misure: queste vengono inviate e quindi registrate con una data periodicità nel sistema. Ogni misura salvata nel sistema deve riportare un timestamp.

PROGETTO AA 2022/23

(variazioni sul tema)

Fase di progettazione

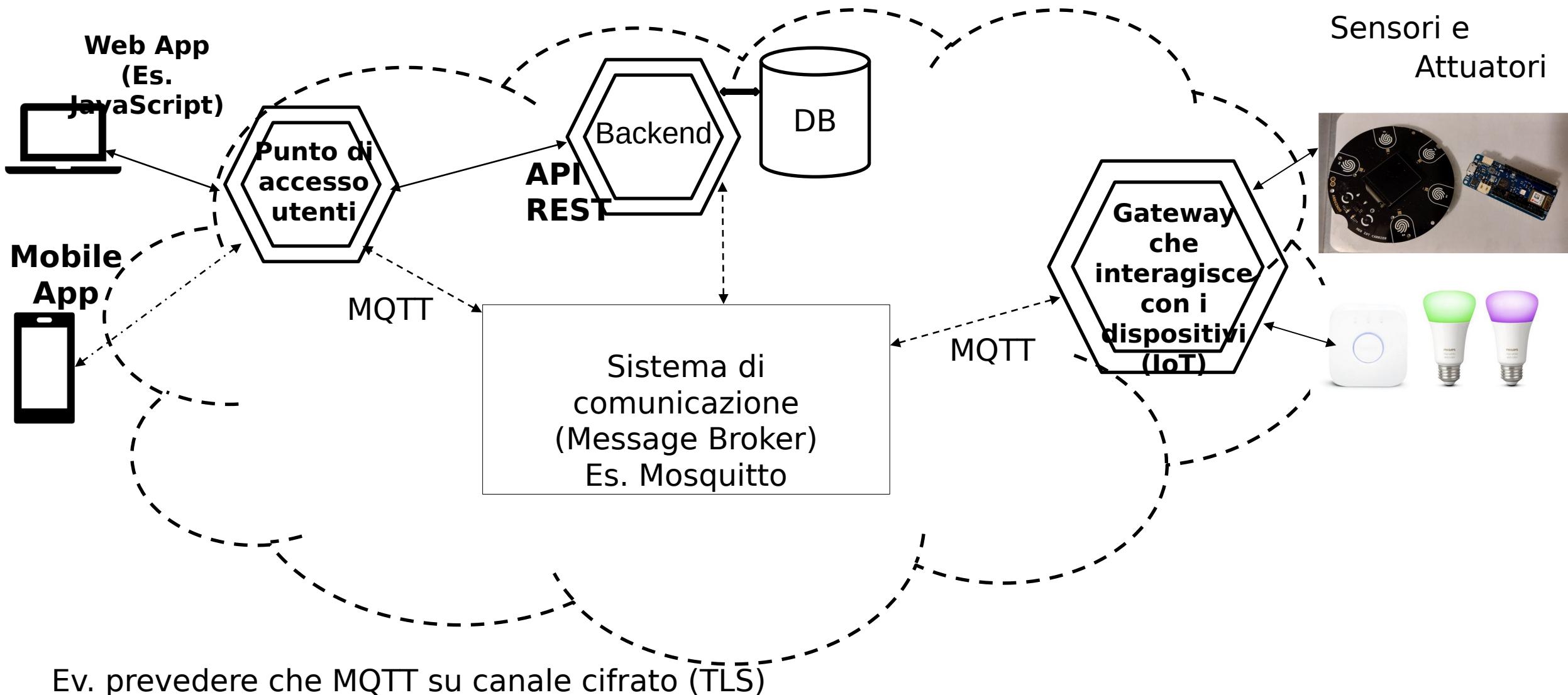
(dopo fase di specifica, precede l'implementazione)

FASE 2: Progettazione

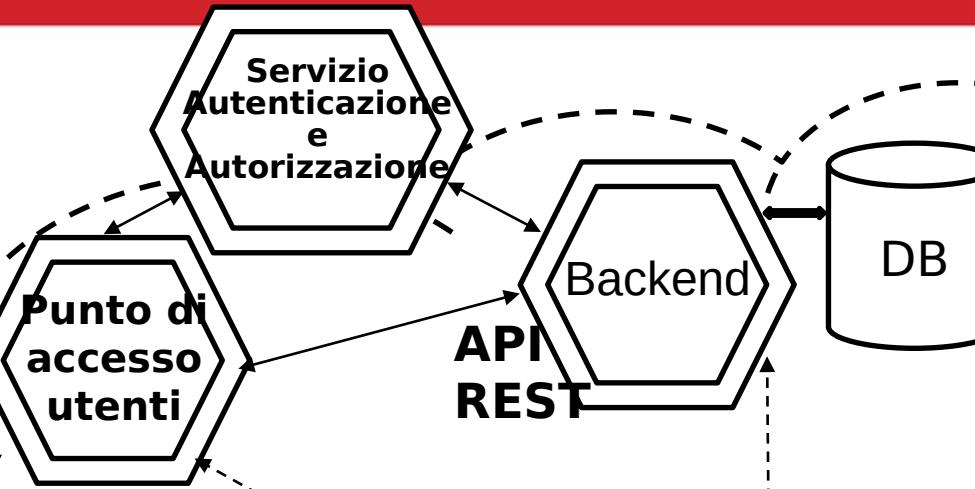
- Nella fase di specifica avete descritto COSA si doveva progettare
- Nella fase di progettazione dovete definire COME realizzare il sistema

Alcuni suggerimenti su come impostare questa fase:

- Architettura basata su servizi il più possibile flessibile rispetto al deployment dei diversi servizi
- Alcuni servizi devono essere disponibili tramite interfaccia REST



**Web App
(Es.
JavaScript)**



Mobile App



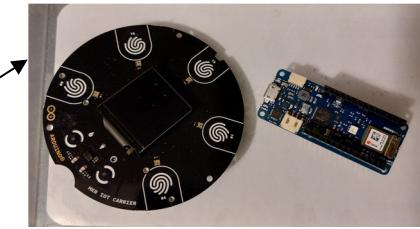
MQTT

Sistema di comunicazione
(Message Broker)
Es. Mosquitto

MQTT

Ev. prevedere che MQTT su canale cifrato (TLS)

Sensori e Attuatori



Gateway che interagisce con i dispositivi (IoT)

Servizio previsioni meteo

Componenti principali dell'architettura:

- Client App: può essere molto semplice (linea di comando) oppure eseguito nel browser (es. realizzato con javascript) oppure un'applicazione mobile: interagisce con un microservizio che funge da punto di accesso al sistema e che quale contatta il backend tramite le API REST.
- Backend: permette agli utenti di accedere per consultare le misure presenti nel DB o inserire / modificare i dati relativi alle varie entità gestite. Espone un'interfaccia REST. Comunica con il sottosistema IoT tramite il broker MQTT.
- DB: database contenente tutte le informazioni necessarie al funzionamento del sistema (es. misure rilevate, informazioni di configurazione, ...). Accessibile solo tramite il backend
- Sottosistema IoT - si occupa di rilevare misure tramite i sensori che invia tramite broker MQTT e tramite lo stesso canale di comunicazione può accettare comandi per gli attuatori. Può essere quindi sia publisher che subscriber.

Cosa produrre nella fase di progettazione

- Per ciascun servizio: diagramma delle classi con i relativi attributi e metodi, diagrammi di sequenza per mostrare le interazioni tra diversi oggetti ed eventualmente diagrammi di attività per mostrare la suddivisione dei compiti tra diversi thread di un certo servizio.
- Può essere utile strutturare in package le diverse classi per evidenziarne i diversi componenti.

Cosa produrre nella fase di progettazione

Progettazione API REST (e documentazione ...)

- Definire Risorse
- Definire la Rappresentazione delle risorse (che verrà scambiato tra client e server; noi useremo un formato JSON)
- Definire gli Endpoint
- Definire le possibili Azioni
- Definire i possibili Errori

Definizione delle Risorse (dal diagramma classi di dominio)

Siamo interessati ad accedere a singole risorse ma anche a collezioni di risorse (per queste ultime usiamo nomi plurali):

- Coltivazioni o Campi
- Sensori/Attuatori (o IoTdevs per accorparli; per distinguerli si può associare a ciascun device un attributo "tipo")
- Utenti (fornitore idrico / agricoltore)

Rappresentazione delle risorse

Coltivazione:

```
{ "id": 14,  
  "id_azienda_agri": 3,  
  "descrizione": "Serra fiori",  
  "fabbisogno": "30",  
  ....,  
  "IoTdevs": [ {"id": 10, "tipo": "sensore_temp"}, {"id": 24, "tipo": "sensore_umid"}, {"id": 28, "tipo": "impianto_irrig1"} ] }
```

Rappresentazione delle risorse

Misurazione:

```
{"id": 1,  
"idColtivazione": 2,  
"data": "2021-05-11",  
"orario": {"ore": 9, "minuti":10},  
"valore":"30"  
}
```

Endpoint

Definire le URI (Uniform Resource Identifier) corrispondenti ai possibili endpoint che permettono di accedere alle risorse; hanno in comune una URI base:

http://api.greenfarms.org - può anche essere utile includere la versione

Esempio: **/v1/aziende** restituisce un array di oggetti "aziende agricole"

Esempio: **/v1/aziende/14** restituisce l'azienda con id 14

Dato che le aziende possiedono coltivazioni e queste contengono dei device IoT possiamo utilizzare una URI gerarchica

Esempio: **/v1/aziende/14/coltivazioni/10**

È anche possibile aggiungere dei parametri definendo così delle query:

Esempio: **/v1/aziende/14/misure?data=2023-05-12**

Nota: per esperimenti in locale la URI base sarà semplicemente localhost:porta

Azioni CRUD sulle risorse

Possibili azioni sulle risorse:

- Visualizzare le coltivazioni di una data azienda agricola
- Creare una nuova coltivazione
- Modificare una coltivazione esistente
- Cancellare una coltivazione

Associamo ciascuna azione ad un «verbo» http – GET, POST, PUT, DELETE; inoltre consideriamo anche il tipo di errore da restituire in caso di fallimento:

Azioni CRUD sulle risorse

Codici Restituiti

- 200 (successo: GET nel body c'è la risorsa, PUT/POST nel body informazioni sull'esito)
- 201 (POST conferma creazione risorsa, restituisce id)
- 400: richiesta non valida (errore sintassi)
- 401: accesso non autorizzato
- 404: risorsa non trovata
- 500: Errore interno

Tabella delle possibili azioni, con eventuale input e risposta					
Verbo http	Endpoint	Input	Output in caso di successo	Messaggio Errore	Descrizione
GET	/aziende/{idAzienda}/coltivazioni/	Body: vuoto	Stato: 200 Body: lista coltivazioni	Stato: 500	Fornisce un array di coltivazioni
GET	/aziende/{idAzienda}/coltivazione/{idColtivazione}	Body: vuoto	Stato: 200 Body: dati coltivazione	Stato: 404 o 500	Fornisce la coltivazione {idColtivazione}
GET	/aziende/{idAzienda}/misure ev. default=data oggi	Body: vuoto Parametro: data (opzionali oraInizio, oraFine)	Stato: 200 Body: lista misure nel giorno (ora)	Stato : 500	Fornisce un array di misure
POST	/richieste_acqua	Body: nuova richiesta	Stato: 201 Body: id della nuova richiesta	Stato: 401, 500	Inserisce nuova richiesta
PUT	/aziende/{id}	Body: nuovi attributi da sostituire	Stato: 200 Body: vuoto	Stato: 401, 404, 500	Modifica dati azienda

Tabella delle possibili azioni, con eventuale input e risposta

Verbo http	Endpoint	Input	Output in caso di successo	Messggio Errore	Descrizione
GET	/aziende/{idAzienda}/ coltivazioni/ {idColtivazione}/ IoTDevs	Body: vuoto	Stato: 200 Body: dati dispositivi presenti nella serra	Stato: 500	Fornisce l'array dei dispositivi della coltivazione {idColtivazione}
DELETE	/aziende/{idAzienda}/ richieste/{idRichiesta}	Body: vuoto	Stato: 200	Stato : 404 o 500	Cancella una richiesta (se esiste)
DELETE	/aziende/{idAzienda}/ richieste	Non Definito	Non Definito	Stato: 400	Azione vietata

Cosa produrre nella fase di progettazione

Progettazione TOPIC e messaggi MQTT

aziendaYYY/coltivazioneXXX/sensori/sensoreTemp oppure
aziendaYYY/coltivazioneXXX /sensoreTempZZZ (o
sensoreUmidWWW)

{"tempCelsius": 18.5, "time": data-e-ora}

{"percUmid": 70, "time": data-e-ora}

(simile per tutti gli altri sensori)

aziendaYYY/serraXXX/attuatori/attuatoreIrrig

{"attivo": true, "time_start": data-e-ora, "time_stop": data-e-ora}

Uso wildcard

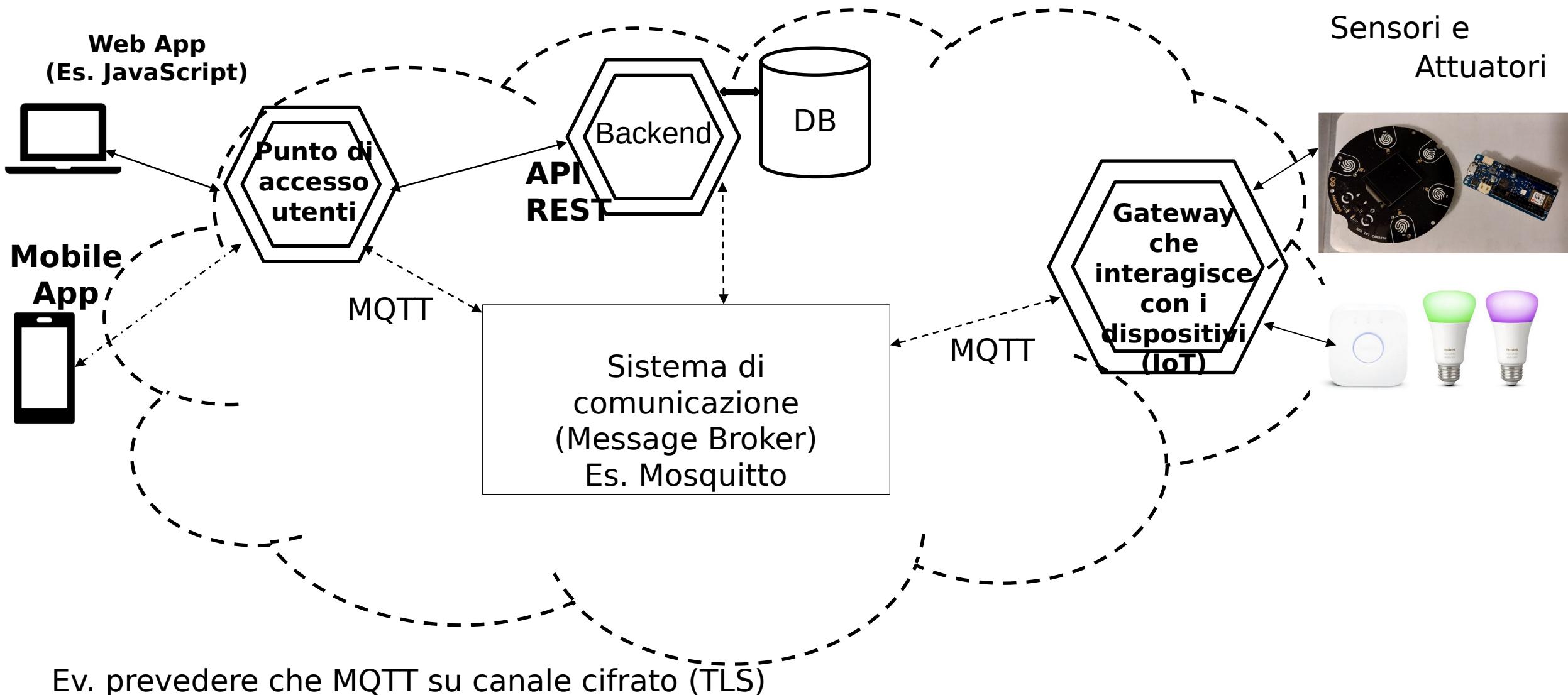
Sottoscrizione alle misure più sensori:

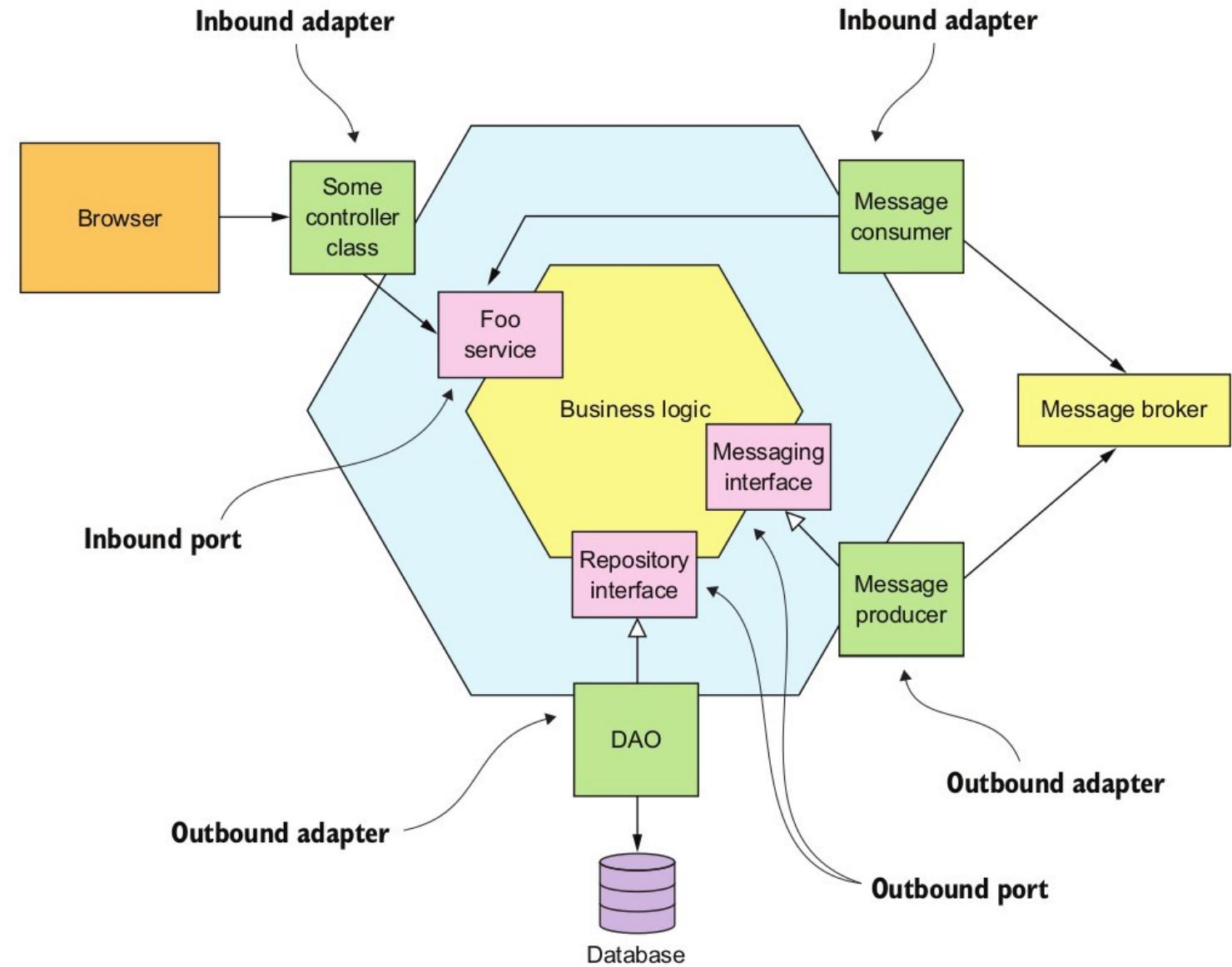
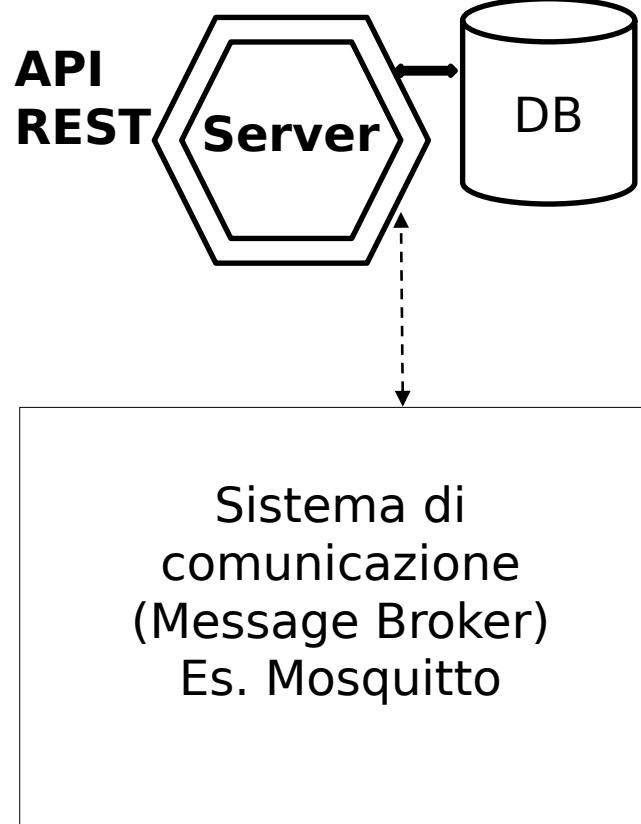
aziendaYYY/serraXXX /sensori/#

Tutti i sensori della serraXXX

aziendaYYY/+sensori/#

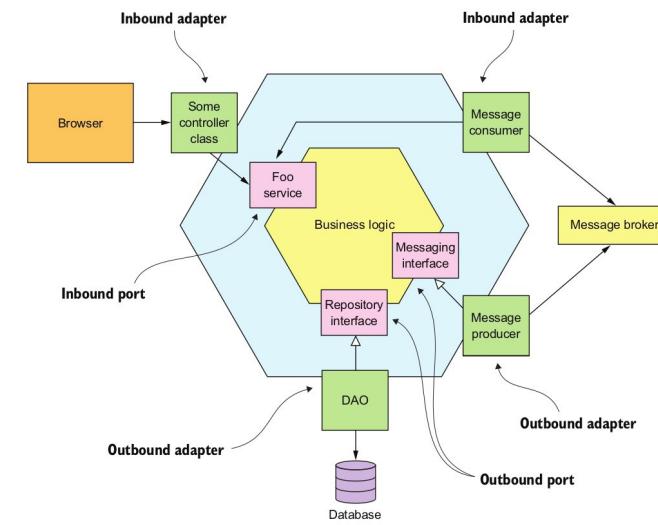
Tutti i sensori di tutte le serre dell'aziendaYYY





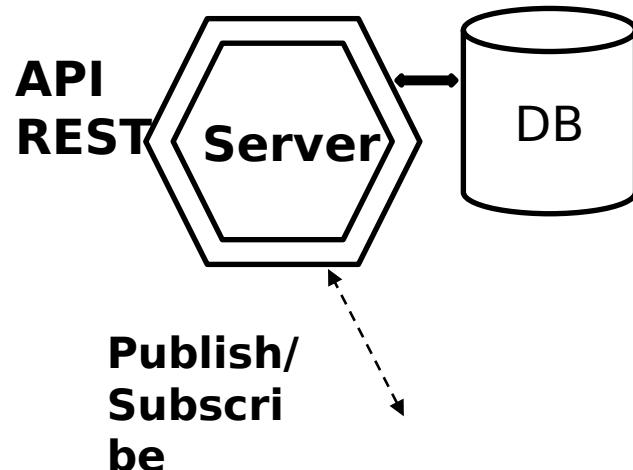
Classi: business logic e adapters

- Interfaccia della business logic: metodi che corrispondono a comandi che cambiano lo stato interno o a query che richiedono risposte
- Adapter
 - Inbound: costituiscono un'interfaccia per il mondo esterno che invia comandi e query al microservizio. Es. API REST
 - Outbound: costituiscono un'interfaccia verso sistemi esterni
 - Database
 - Broker



API REST ADAPTER (Inbound Adapter)

Classi che si occupano di gestire le richieste in arrivo e richiamano i metodi offerti dal cuore del microservizio



Business Logic (gestore serre, piani di irrigazione o illuminazione e misure)

Classi che si occupano di rispondere a comandi/query:

- crea prenotazione, cancella prenotazione, modifica prenotazione, elenco prenotazioni
- identifica irrigatore attuale, notifica inizio/fine irrigazione
- registra misura, elenco misure

PERSISTENZA
(outbound adapter)
Mappa operazioni CRUD sugli oggetti trattati dalla business logic su operazioni sul DB

Broker messaging ADAPTER (In/Outbound Adapter)

- In: riceve misure (via subscribe a broker) – richiama regista misura
- Out: invia notifiche di inizio/fine irrigazione – consuma notifiche



View (UI)

Classi che si occupano di gestire le viste dell'interfaccia utente

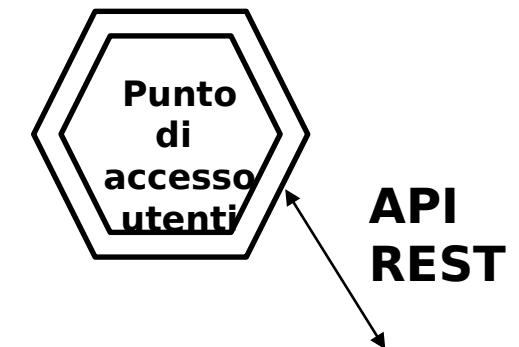
Controller

gestisce la sequenza di interazioni con l'utente (innescate dalla UI) e di conseguenza aggiorna il modello

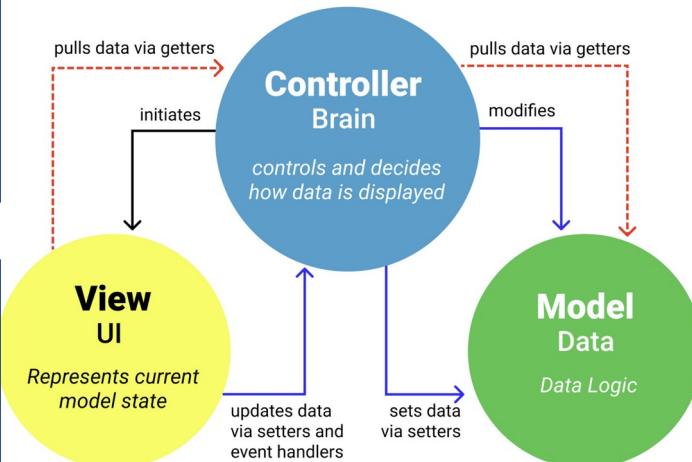
Model (gestisce gli oggetti del dominio)

- Interagisce con l'outbound adapter
- Aggiorna le viste

Outbound Adapter verso il Servizio che gestisce coltivazioni e misure



MVC Architecture Pattern



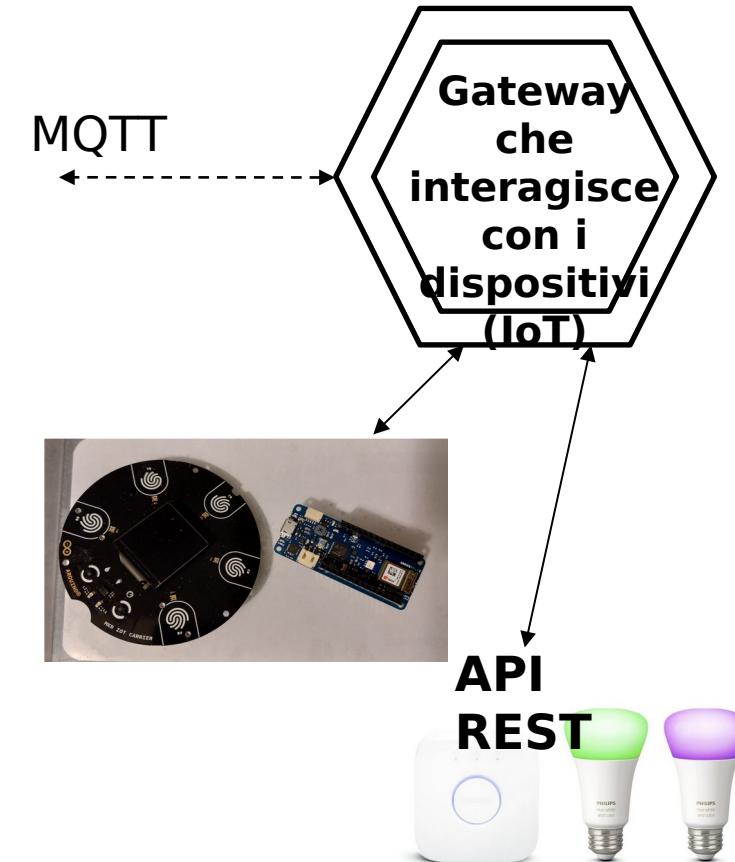
IoT ADAPTER (In/Outbound Adapter)

- In: rilevare misure (attraverso il meccanismo di interazione con sensori)
- Out: comanda gli attuatori (es. richiama le API REST delle lampadine)

Gestisce una tabella serre e sensori e i piani di irrigazione in corso
Gestisce le informazioni su misure monitorabili e sullo stato di attuatori
Monitorare i parametri di stato delle serre
Agisce per mantenerli modificando lo stato degli attuatori

Broker messaging ADAPTER (In/Outbound Adapter)

- Out: invia misure (via publish a broker) – richiama lettura sensori
- In: riceve notifiche di inizio/fine irrigazione – comanda gli attuatori



Checklist fase specifica:

CASI D'USO

Descrizione tramite diagramma UML e/o testuale dei casi d'uso previsti: dovranno rappresentare in modo sufficientemente chiaro COSA il sistema dovrà fare. Se usate la rappresentazione UML aggiungete annotazioni e/o un testo di accompagnamento per dettagliare.

REQUISITI FUNZIONALI E NON FUNZIONALI

Dalla descrizione dei casi d'uso potrete ricavare i requisiti funzionali.

Può essere utile aggiungere anche i requisiti non funzionali che possono comprendere vincoli da tenere in conto nella successiva fase di progettazione: per esempio vincoli sulla infrastruttura utilizzata per mettere in comunicazione i diversi elementi dell'architettura (uso di MQTT per comunicare con il sottosistema IoT), linguaggio ed eventuali framework scelti per l'implementazione.

DIAGRAMMA DELLE CLASSI DEL DOMINIO

Il diagramma delle classi del dominio descrive le entità gestite dall'applicazione e le relazioni tra tali entità. In questo diagramma non si fa ancora riferimento all'architettura del sistema da implementare, e non è necessario elencare i metodi delle classi. Anche in questo caso le annotazioni possono aiutare a comprendere meglio il diagramma.

Checklist fase di progettazione

Diagramma delle classi: questa volta si tratta delle classi che saranno implementate. Ogni microservizio dell'architettura avrà le sue classi. Potete rappresentare una relazione tra classi di microservizi diversi se queste sono responsabili della comunicazione tra microservizi: in questo caso si può aggiungere una annotazione che indica in che modo dovrà avvenire la comunicazione (es. via API REST oppure via MQTT, mediata da un broker).

Per le interazioni un po' articolate tra classi (dello stesso microservizio o di microservizi diversi) realizzare un diagramma di sequenza.

Se alcune classi hanno un comportamento complesso che ritenete valga la pena dettagliare potete usare i diagrammi di attività o di stato.

Descrizione delle API REST e dei TOPIC MQTT + formato messaggi. Volendo potete rappresentare i topic MQTT in forma di albero per evidenziare la struttura gerarchica: aggiungete degli esempi su come si possono sfruttare le wildcard (+, #) nei topic per sottoscrivere più topic contemporaneamente.

Checklist fase di implementazione

Dovrete consegnare:

- il codice (via gitlab) ben commentato;
- il dump del database;
- le istruzioni per l'installazione e l'avvio dell'applicazione: queste potreste inserirle nel README.md;
- l'indicazione di come vi siete suddivisi i compiti nel gruppo.

NOTA: anche se avete collaborato un po' su tutto individuate uno/due referenti per ciascun microservizio – durante il colloquio rivolgeremo ai referenti domande più dettagliate sull'implementazione del microservizio di loro competenza.

SEMPLIFICAZIONI ACCETTABILI:

- Va bene considerare un solo fornitore di risorse idriche (anche se nel DB dovrebbe esserci una tabella dove eventualmente in prospettiva se ne potrebbero aggiungere altri: all'unico fornitore presente in tabella si potrà agganciare la tabella che conterrà le richieste accettate di fornitura d'acqua giornaliera, dove si potrà anche memorizzare giorno per giorno il consumo effettivo). Quindi nell'implementazione potete considerarne un solo fornitore d'acqua, e non prevedere l'indicazione di QUALE fornitore dev'essere il destinatario delle richieste di risorse idriche (sarà automaticamente impostato per default l'unico presente).
- La quantità complessiva d'acqua disponibile (lato fornitore) potrebbe variare nel tempo, ma se questa si riduce al di sotto delle richieste già soddisfatte potrebbe essere necessario rivedere le quantità già concesse agli agricoltori: se per semplicità si decide che la quantità complessiva non possa ridursi al di sotto di quanto già concesso, in modo da non creare il problema della revisione di quanto già concordato, può andare bene (soprattutto per i gruppi meno numerosi questa scelta è utile per semplificare l'implementazione). Chiaramente in questo caso dovrà essere fatto un controllo sulla richiesta dell'utente "fornitore" di fare modifiche sui dati di disponibilità d'acqua, per non consentire una riduzione incompatibile con gli accordi già presi con gli agricoltori. Eventualmente in fase di demo del vostro sistema, nel caso si volesse provare a ridurre, ci preoccuperemo di ridurre prima qualche richiesta di qualche agricoltore per rendere la riduzione possibile. Non ci sono invece problemi se si aumenta la disponibilità.

ALCUNE INDICAZIONI:

- Per poter fare un po' di test sull'applicazione bisognerà trovare il modo di far scorrere il tempo più velocemente. A questo scopo possiamo suddividere la "giornata virtuale" in fasi (e possiamo inventare qualche meccanismo – anche artigianale – per segnare l'inizio di ogni nuova fase in modo da poter sincronizzare tutto il sistema su tali eventi – potrebbe essere l'invio di un messaggio MQTT):
 - Iniziamo dalla "sera" della giornata N: questa è la fase dell'assegnazione d'acqua per il giorno successivo; in questa fase idealmente gli agricoltori, dopo aver controllato i consumi effettivi della giornata, possono aggiornare la quantità richiesta per la giornata successiva (o confermare quella del giorno precedente – anche implicitamente, se non si intende avanzare alcuna nuova richiesta). Il fornitore potrà o meno accordare le nuove quantità richieste (in caso di rifiuto il sistema potrebbe chiedere se si vuole mantenere la quantità già pattuita o modificare la richiesta). E' sempre in questa fase che gli agricoltori potranno eventualmente modificare la suddivisione delle risorse disponibili tra i propri campi.
 - Ora siamo al "mattino" della giornata N+1: all'inizio di questa fase essendo in una nuova giornata si "resettano" le scorte e l'irrigazione riparte con le scorte rinnovate. I sensori misurano l'umidità e se necessario gli attuatori iniziano l'irrigazione: in questa fase si può far scorrere il tempo virtuale ad una velocità definita, e quindi calcolare la quantità d'acqua già consumata in ogni porzione di tempo in base allo stato dell'attuatore (naturalmente quando l'acqua destinata a quel campo si esaurisce si interrompe l'irrigazione: questo evento si potrà registrare nel DB per indicare che tutta l'acqua disponibile è stata consumata prima della fine della giornata).
 - Durante tutta la giornata vengono registrate periodicamente nel DB le misure di umidità e temperatura in ogni campo ed eventualmente i cambiamenti di stato degli attuatori (irrigatore acceso/spento). Sarà il gestore IoT a raccogliere le misure (che per la demo potrebbero essere prodotte in base ad una traccia artificiale, se non si dispone di un sensore vero) e ad inviarle tramite il protocollo MQTT con cadenza regolare – queste verranno lette dal backend per la registrazione nel DB. Lo stesso gestore si preoccuperà di attivare gli

attuatori o stopparli a seconda delle misure raccolte; i vari cambiamenti di stato potranno essere poi inviati via MQTT al backend unitamente ad aggiornamenti su quanta acqua è già stata consumata. Per sapere quali sono le esigenze di umidità del campo, la quantità d'acqua disponibile, o altri parametri sul sistema di irrigazione o sull'estensione del campo, queste potrebbero essere inviate dal backend (e raccolte dal gestore IoT), sempre via MQTT, ad ogni inizio giornata, o letto tramite chiamata REST inviata dal gestore IoT al backend, quando il gestore IoT si avvia e ad ogni inizio di nuova giornata.

Il Gestore IoT è un eseguibile a sé stante che svolge le seguenti operazioni:

Potremmo avere un oggetto “campo” che contiene un oggetto sense_umid e un oggetto att_irrig, inoltre ha un oggetto che salva il livello di scorta d’acqua per la giornata e quella attualmente disponibile (o quella già consumata). Espone metodi come “getUmidita” o “setStatoAtt”

sense_umid è un oggetto capace di dialogare con i veri sensori, se li abbiamo, oppure un oggetto che emula un sensore se non abbiamo quello reale. att_irrig è in grado di comunicare con l’attuatore reale, oppure emula l’attuatore mantenendo solo in memoria lo stato attuale.

while (true)

```
{      if <<inizio_giornata>> then reset clock e rinnova scorte acqua // prima del reset potremmo registrare se si è avanzata acqua o se c'è stata carenza. Il rinnovo delle scorte può avvenire in base ad una interrogazione fatta al backend oppure ricevendo un messaggio su MQTT
```

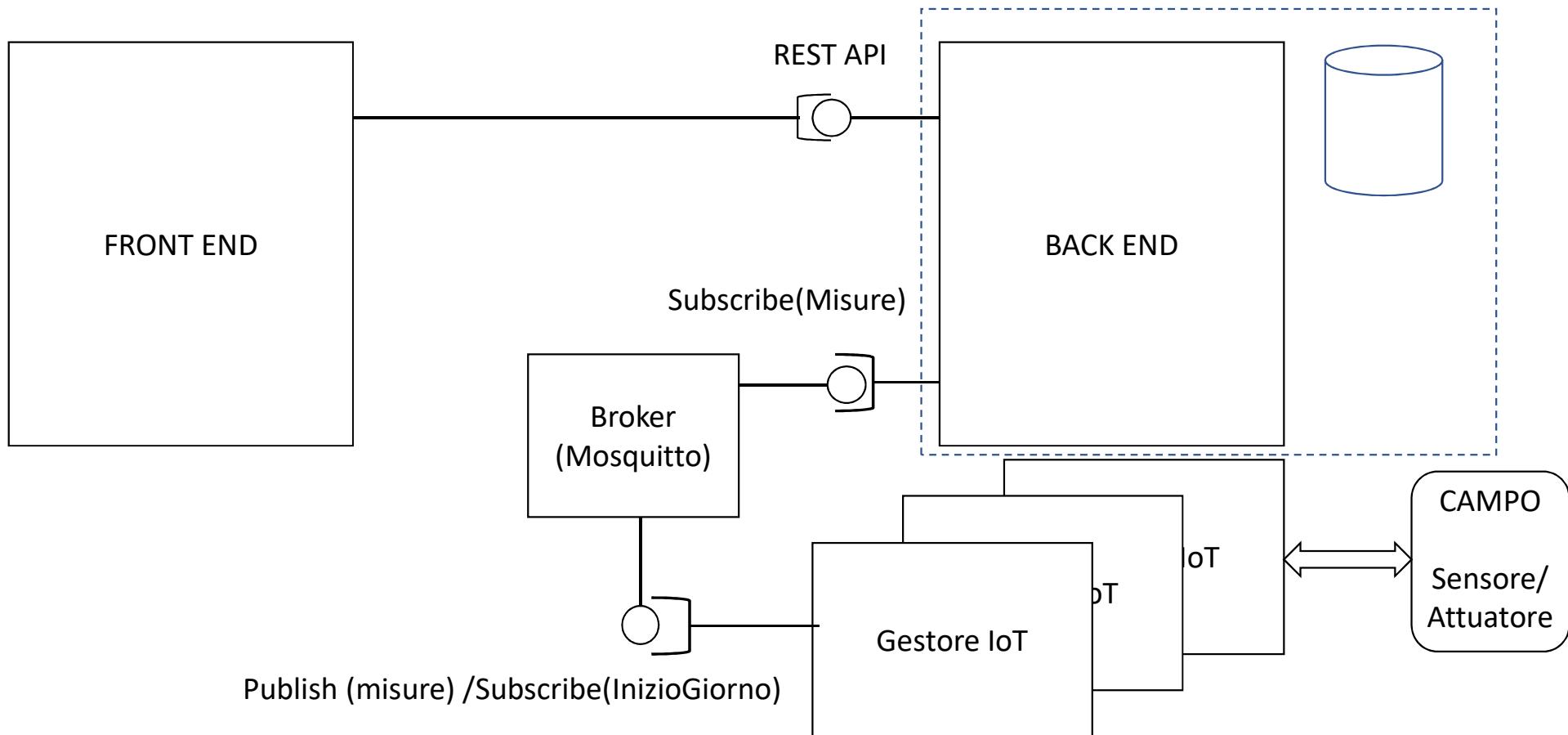
```
    valUmid = campo.getUmidita(); // eventualmente da ripetere su un array di campi se il Gestore IoT ne gestisce più di uno
```

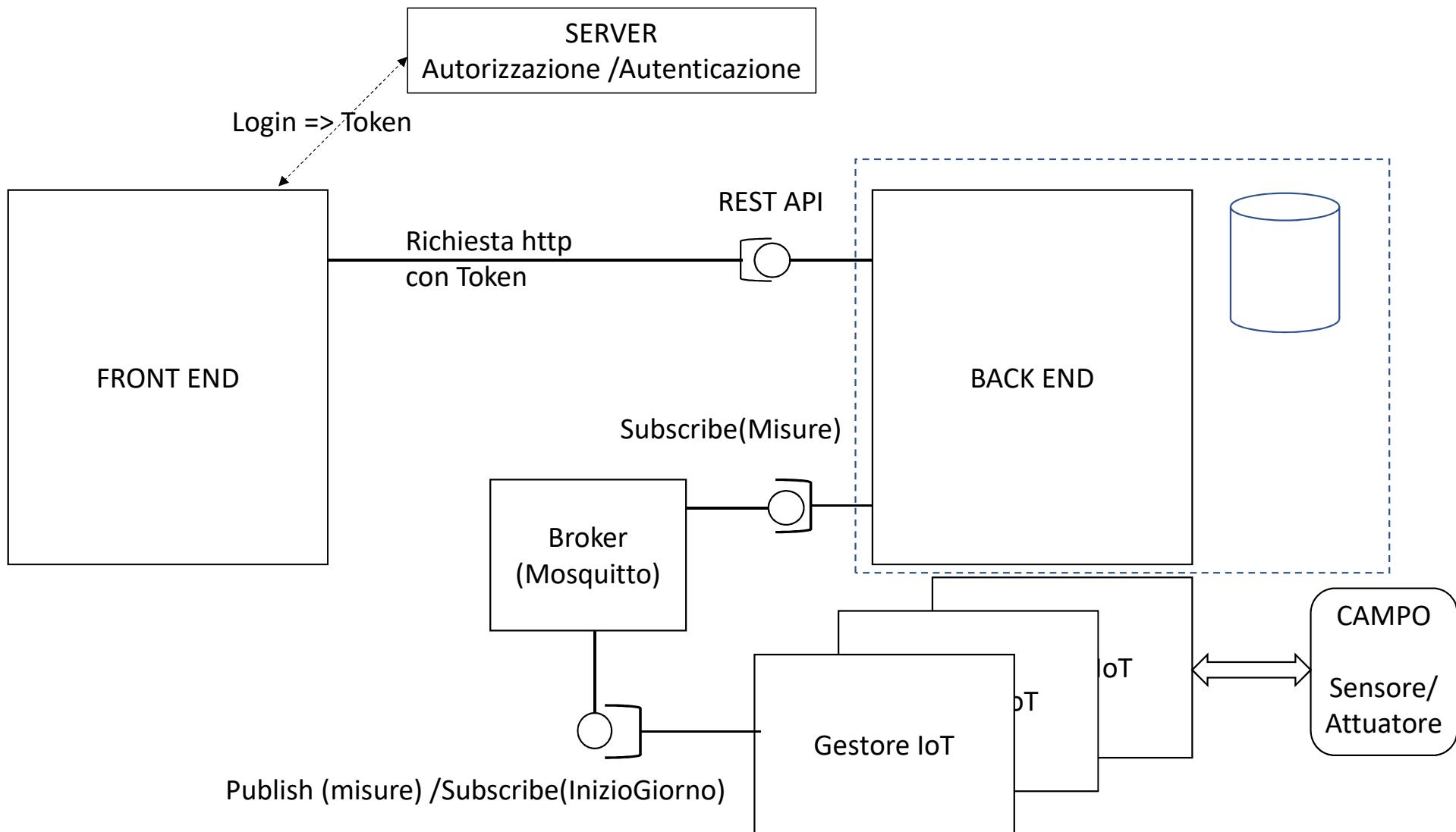
```
    invia al broker su topic farmAPI/<IDcampo>/sensUmid il messaggio contenente {clock, valUmid }  
    newStatoAtt = f(valUmid,altri parametri)  
    if newStatoAtt <> campo.getStatoAtt() // è cambiato lo stato  
        campo.setStatoAtt(newStatoAtt)  
        invia al broker su topic farmAPI/<IDcampo>/statoAtt il messaggio contenente {clock,  
newStatoAtt }  
        campo.aggiornaScortaAcqua; // se l'attuatore è ON la scorta viene decrementata di un tot da definire, altrimenti rimane invariata  
        clock++;  
        sleep(DELTA T) // passo di avanzamento del clock  
}
```

Nota: quando leggiamo l’umidità, se il metodo è implementato da un emulatore, dovremmo con qualche criterio anche aggiornarla (es. se attuatore ON o piove l’umidità aumenta di un tot, se OFF e piove aumenta un po’ meno, se OFF e c’è sole diminuisce), se invece è implementato da una classe capace di interagire con i veri sensori il valore restituito sarà quello reale letto dai sensori e se analogamente l’attuatore questo sarà comandato da una classe capace di interagire con l’attuatore per accenderlo e spegnerlo.

Nota: la funzione f(valUmid,altri parametri) racchiude il criterio di controllo degli attuatori: dovrà anche tenere conto della riserva d’acqua disponibile. Eventualmente possiamo usare un terzo stato BLOCK per indicare che l’attuatore è bloccato in attesa che vengano ripristinate le scorte d’acqua.

L’inizio giornata è impostato per esempio tramite un messaggio su un topic farmAPI/segnaleOrario

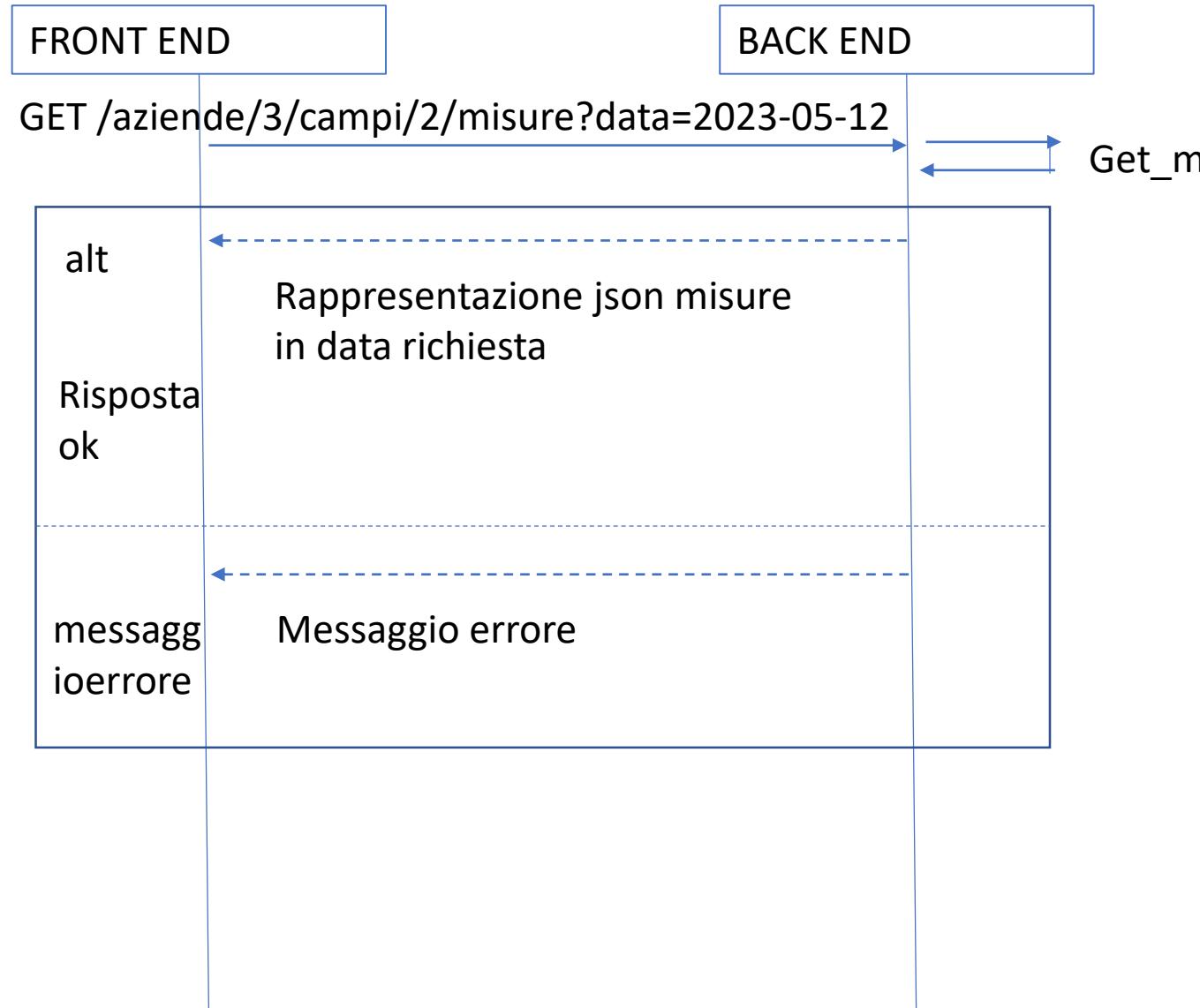


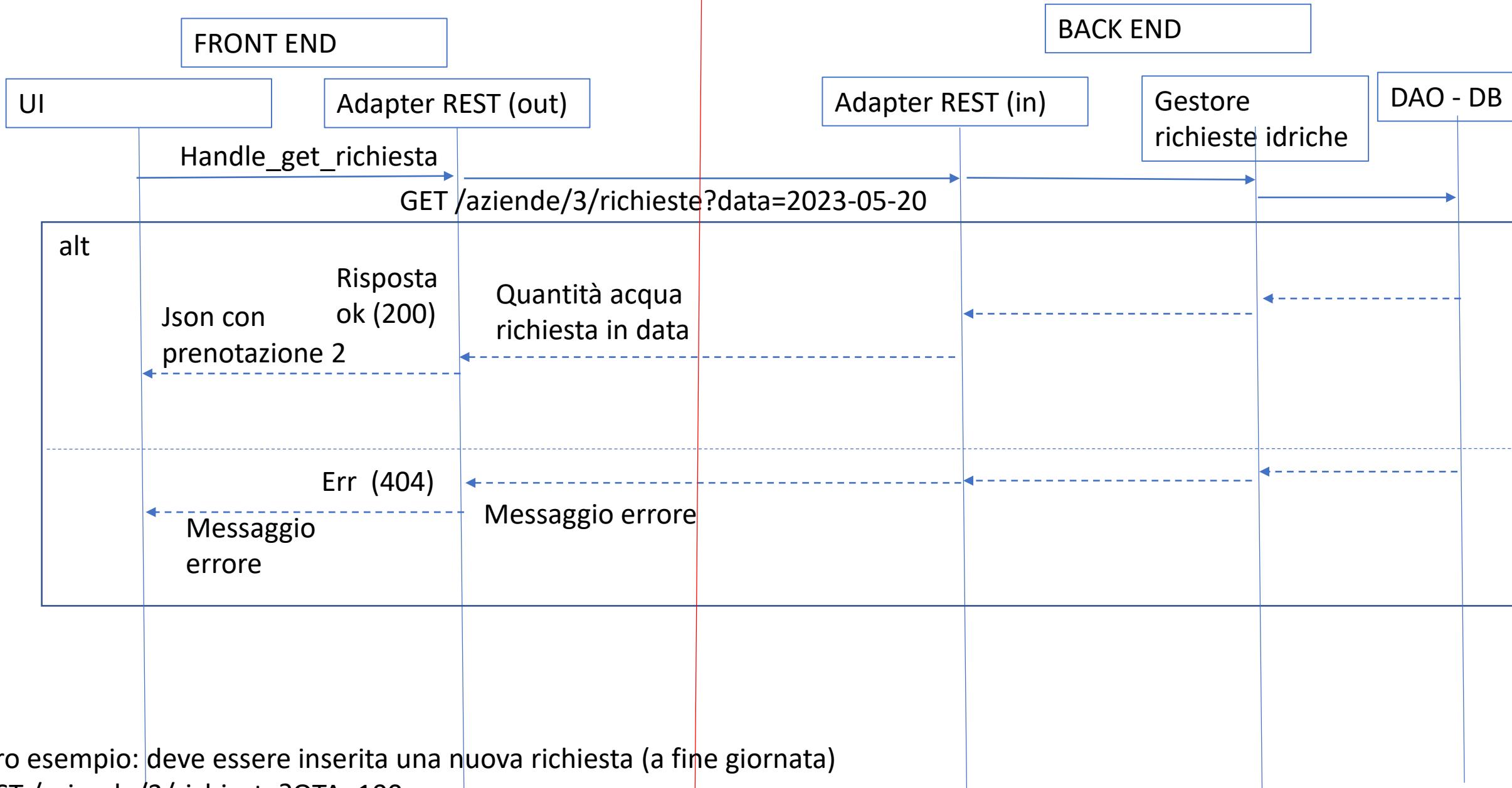


Diagrammi di sequenza

Interazione tra componenti

Interazione tra oggetti (interni ai componenti)





Altro esempio: deve essere inserita una nuova richiesta (a fine giornata)

POST /aziende/3/richieste?QTA=100

Può essere accettata (restituisce l'ID della richiesta) oppure rifiutata (non può essere soddisfatta).

Progettazione e Documentazione API REST

Servizio Backend

- Il servizio messo a disposizione dal backend è disponibile tramite interfaccia di tipo API REST
- La documentazione del progetto deve prevedere anche la parte relativa alla specifica delle API REST: per ogni endpoint il tipo di operazione che si può compiere, formato di eventuali parametri e dati da inviare nel body e formato eventuale body della risposta.
- Uno strumento che potete utilmente impiegare nella fase di progettazione delle API ma anche per documentarle è lo Swagger Editor (che implementa Open API)

Tutorial Open API

<https://support.smartbear.com/swaggerhub/docs/tutorials/openapi-3-tutorial.html>

Il tutorial qui indicato mostra un esempio di definizione di API REST per un semplice servizio, introducendo gli aspetti essenziali della specifica OpenAPI 3.0, un formato che è diventato uno standard de-facto per documentare API REST.

È possibile utilizzare un editor on-line per definire un insieme di API REST, e salvarlo in un file di testo in formato YAML (che si può ricaricare nell'editor successivamente per prenderne visione o perfezionarlo)

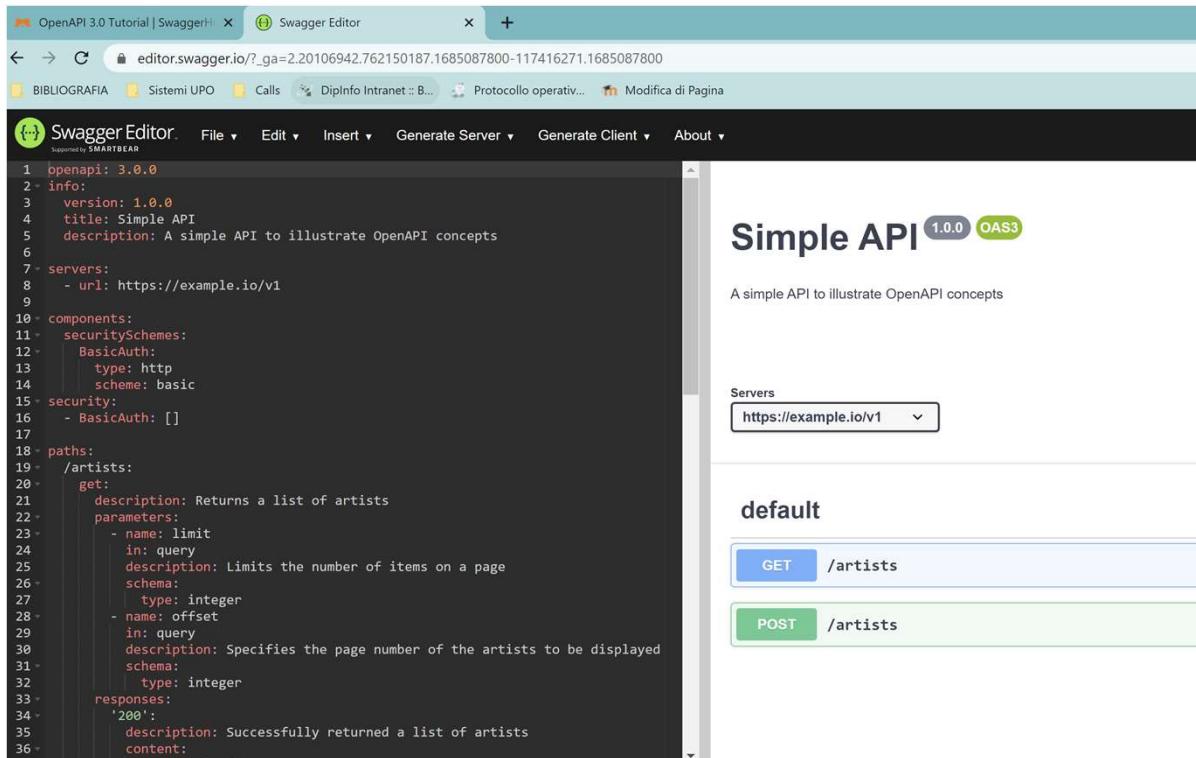
<https://swagger.io/tools/swagger-editor/>

È anche possibile utilizzare JSON per la rappresentazione (anche se è consigliato YAML)

YAML YAML Ain't Markup Language <https://yaml.org/spec/1.2.2>

- It is a data serialization language designed to be human-friendly and work well with modern programming languages for common everyday tasks.
- YAML was specifically created to work well for common use cases such as: configuration files, log files, interprocess messaging, cross-language data sharing, object persistence and debugging of complex data structures.
- It uses Unicode [printable](#) characters, [some](#) of which provide structural information and the rest containing the data itself. YAML achieves a unique cleanliness by minimizing the amount of structural characters and allowing the data to show itself in a natural and meaningful way.

L'editor on-line



The screenshot shows the Swagger Editor interface. On the left, the OpenAPI 3.0 specification is displayed in a code editor:

```
1 openapi: 3.0.0
2 info:
3   version: 1.0.0
4   title: Simple API
5   description: A simple API to illustrate OpenAPI concepts
6
7 servers:
8   - url: https://example.io/v1
9
10 components:
11   securitySchemes:
12     BasicAuth:
13       type: http
14       scheme: basic
15   security:
16     - BasicAuth: []
17
18 paths:
19   /artists:
20     get:
21       description: Returns a list of artists
22       parameters:
23         - name: limit
24           in: query
25           description: Limits the number of items on a page
26           schema:
27             type: integer
28         - name: offset
29           in: query
30           description: Specifies the page number of the artists to be displayed
31           schema:
32             type: integer
33       responses:
34         '200':
35           description: Successfully returned a list of artists
36           content:
```

On the right, the generated API documentation is shown under the heading "Simple API 1.0.0 OAS3". It includes a brief description: "A simple API to illustrate OpenAPI concepts", a dropdown for "Servers" set to "https://example.io/v1", and a "default" section with two operations: a "GET /artists" operation and a "POST /artists" operation.

Le informazioni contenute in una specifica

- Meta information
- Path items (endpoints):
 - Parameters
 - Request bodies
 - Responses
- Reusable components:
 - Schemas (data models)
 - Parameters
 - Responses
 - Other components

Meta informazione

Versione di OpenAPI,
Versione API, Titolo,
descrizione, base URL e
altre informazioni
generali

```
openapi: 3.0.0
info:
  version: 1.0.0
  title: Simple Artist API
  description: A simple API to illustrate OpenAPI concepts

servers:
  - url: https://example.io/v1

# Basic authentication
components:
  securitySchemes:
    BasicAuth:
      type: http
      scheme: basic
  security:
    - BasicAuth: []
```

Sezione dei path

In questa sezione si descrivono gli endpoint e le azioni (verbi HTTP) che si possono compiere sulle risorse corrispondenti.

Viene anche specificata la forma delle possibili risposte

```
paths:  
  /artists:  
    get:  
      description: Returns a list of artists  
      # ----- Added lines -----  
      responses:  
        '200':  
          description: Successfully returned a list of artists  
          content:  
            application/json:  
              schema:  
                type: array  
                items:  
                  type: object  
                  required:  
                    - username  
                  properties:  
                    artist_name:  
                      type: string  
                    artist_genre:  
                      type: string  
                    albums_recorded:  
                      type: integer  
                    username:  
                      type: string
```

Sezione dei path

Viene anche specificata la forma delle possibili risposte, incluse le segnalazioni di errore

```
'400':  
    description: Invalid request  
    content:  
        application/json:  
            schema:  
                type: object  
                properties:  
                    message:  
                        type: string
```

GET https://example.io/v1/artists?limit=20&offset=3



Si possono specificare parametri per gli endpoint

```
paths:  
  /artists:  
    get:  
      description: Returns a list of artists  
      # ----- Added lines -----  
      parameters:  
        - name: limit  
          in: query  
          description: Limits the number of items on a page  
          schema:  
            type: integer  
        - name: offset  
          in: query  
          description: Specifies the page number of the artists to be displayed  
          schema:  
            type: integer  
      # ----- /Added lines -----  
      responses:
```

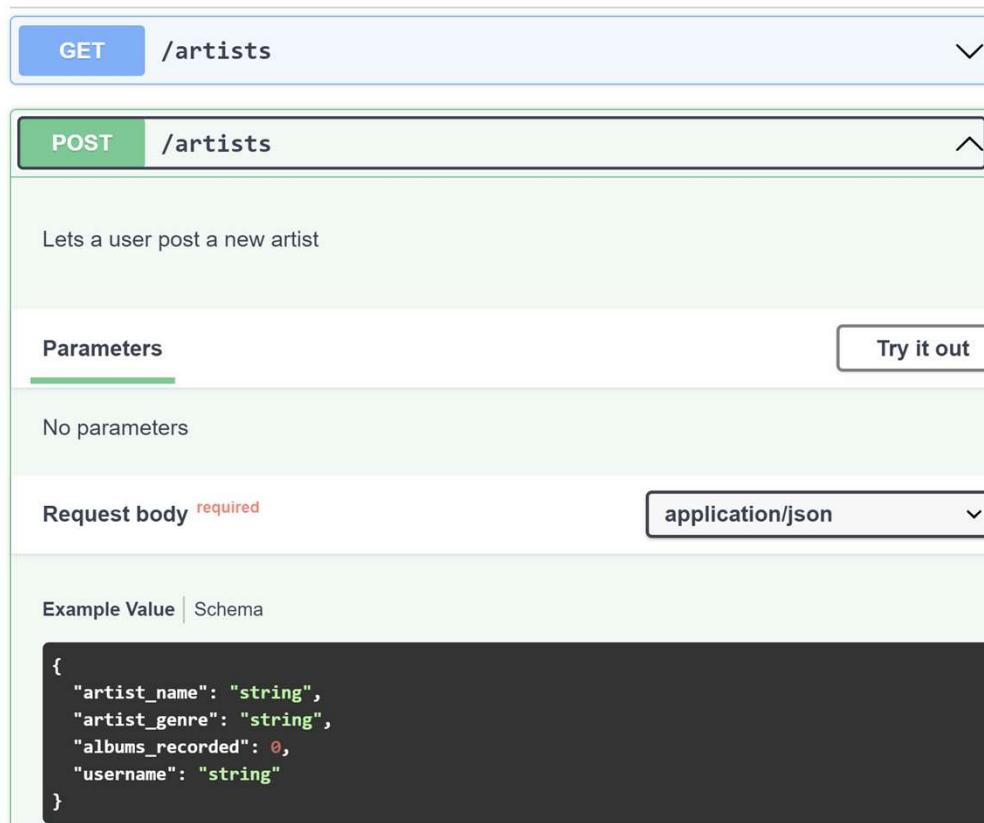
Parameters	
Name	Description
limit	Limits the number of items on a page integer (query) limit
offset	Specifies the page number of the artists to be displayed integer (query) offset

GET https://example.io/v1/artists?limit=20&offset=3

POST

```
----- ADDED LINES -----  
post:  
    description: Lets a user post a new artist  
    requestBody:  
        required: true  
        content:  
            application/json:  
                schema:  
                    type: object  
                    required:  
                        - username  
                    properties:  
                        artist_name:  
                            type: string  
                        artist_genre:  
                            type: string  
                        albums_recorded:  
                            type: integer  
                        username:  
                            type: string  
  
responses:  
    '200':  
        description: Successfully created a new artist  
  
    '400':  
        description: Invalid request  
        content:
```

Visualizzazione degli endpoint specificati



The screenshot shows a detailed view of an API endpoint for artists. At the top, there is a blue button labeled "GET" followed by the endpoint path "/artists". Below this, there is a green button labeled "POST" also followed by the endpoint path "/artists". A descriptive text below the POST method states: "Lets a user post a new artist". Under the "Parameters" section, it says "No parameters". In the "Request body" section, the word "required" is highlighted in red, and the media type "application/json" is selected. The "Example Value" tab is active, showing a JSON schema:

```
{  
    "artist_name": "string",  
    "artist_genre": "string",  
    "albums_recorded": 0,  
    "username": "string"  
}
```

I parametri possono anche essere all'interno del path

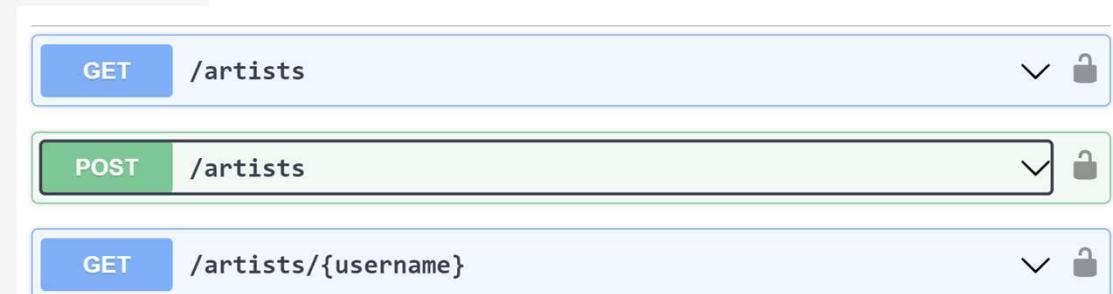
```
/artists/{username}:
get:
  description: Obtain information about an artist from his or her unique username
  parameters:
    - name: username
      in: path
      required: true
      schema:
        type: string

  responses:
    '200':
      description: Successfully returned an artist
      content:
        application/json:
          schema:
            type: object
            properties:
              artist_name:
                type: string
              artist_genre:
                type: string
              albums_recorded:
                type: integer

    '400':
      description: Invalid request
      content:
        application/json:
```

/artists/{username}

In questo caso la risposta non
è un array di item ma è un
singolo item.



The screenshot shows a Swagger UI interface with the following endpoints:

- GET /artists**: A blue button labeled "GET" next to the path "/artists". There is a dropdown arrow and a lock icon to the right.
- POST /artists**: A green button labeled "POST" next to the path "/artists". There is a dropdown arrow and a lock icon to the right.
- GET /artists/{username}**: A blue button labeled "GET" next to the path "/artists/{username}". There is a dropdown arrow and a lock icon to the right.

Per evitare ridondanza: componenti riutilizzabili

- Schemas (data models) ← Esempio gli attributi di ogni «artist»
- Parameters ← Esempio limit - page
- Request bodies
- Responses ← Esempio risposta errore 400
- Response headers
- Examples
- Links
- Callbacks

Sezione components e riferimenti

```
parameters:  
  # ----- Added line -----  
  - $ref: '#/components/parameters/PageLimit'  
  - $ref: '#/components/parameters/PageOffset'  
  # ----- /Added line -----  
responses:  
  '200':
```

```
content:  
  application/json:  
    schema:  
      # ----- Added line -----  
      $ref: '#/components/schemas/Artist'  
      # ----- /Added line -----
```

```
responses:  
  '200':  
    description: Successfully created a new artist  
  '400':  
    # ----- Added line -----  
    $ref: '#/components/responses/400Error'  
    # ----- /Added line -----
```

```
components:  
  securitySchemes:  
    BasicAuth:  
      type: http  
      scheme: basic  
  
schemas:  
  Artist:  
    type: object  
    required:  
      - username  
    properties:  
      artist_name:  
        type: string  
      artist_genre:  
        type: string  
      albums_recorded:  
        type: integer  
      username:  
        type: string  
  
    # ----- Added lines -----  
parameters:  
  PageLimit:  
    name: limit  
    in: query  
    description: Limits the number of items on a page  
    schema:  
      type: integer
```

- Vedere file YAML allegato ... è solo un esempio, dovete ideare voi le API, i parametri, i formati dei dati scambiati, ...

```
1  openapi: 3.0.0
2  info:
3      version: 1.0.0
4      title: Gestione risorse idriche
5      description: API del backend del sistema di gestione risorse idriche
6
7  servers:
8      - url: http://api.greenfarms.org/v1
9
10 paths:
11     /aziende:
12         get:
13             description: Returns a list of companies
14             responses:
15                 '200':
16                     description: Successfully returned a measure
17                     content:
18                         application/json:
19                             schema:
20                                 type: array
21                                 items:
22                                     $ref: '#/components/schemas/Company'
23                 '400':
24                     $ref: '#/components/responses/400Error'
25
26
27     /aziende/{idAzienda}/coltivazioni/{idColtivazione}/misure/{idMisura}:
28         get:
29             description: Returns a measure
30             parameters:
```

API del backend del sistema di gestione risorse idriche

Servers

<http://api.greenfarms.org/v1> ▾

default

GET	/aziende
GET	/aziende/{idAzienda}
GET	/coltivazioni/{idColtivazione}
GET	/misure/{idMisura}

Schemas

Schemi ripetuti:

```
58  components:
59    schemas:
60      Company:
61        type: object
62        required:
63          - idCompany
64          - name
65          - numFields
66        properties:
67          idCompany:
68            type: integer
69          name:
70            type: string
71          numFields:
72            type: integer
73
74      Misurazione:
75        type: object
76        required:
77          - idColtivazione
78          - data
79          - orario
80          - valore
81        properties:
82          id:
83            type: integer
84          idColtivazione:
85            type: integer
86          data:
```

Schemas

```
Company ▾ {
  idCompany*           integer
  name*                string
  numFields*           integer
}

Misurazione ▾ {
  id                   integer
  idColtivazione*     integer
  data*               string
  pattern: ^\d{4}(0[1-9]|1[012])(0[1-9]|1[2][0-9]|3[01])$ example: 20210130 Data rilevazione
  orario*              ▾ {
    ora                 string
    pattern: (?:[0-1]\d|[2][0-3]) ora da 00 a 23
  }
}
```

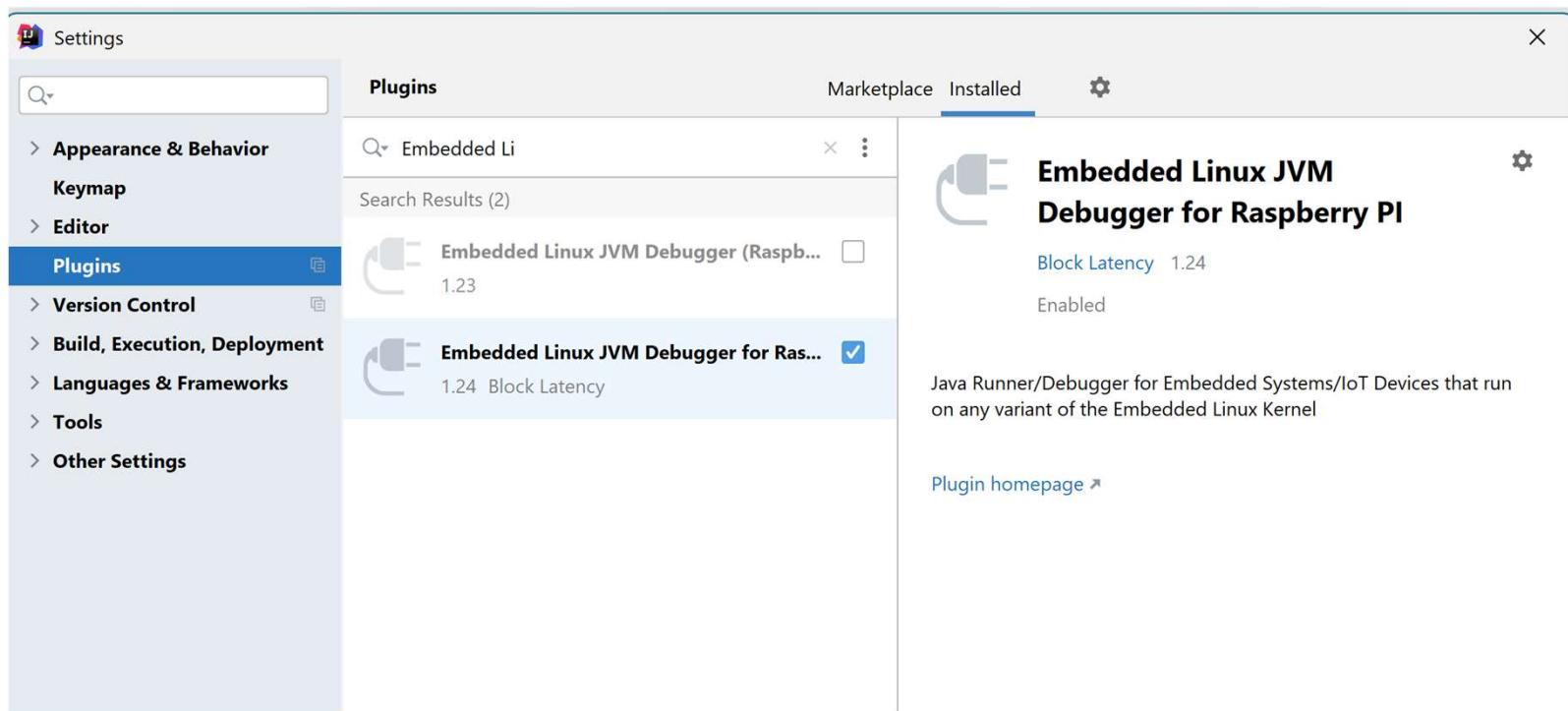
Installare componenti su
Raspberry Pi

Servizi ... o componenti del sistema

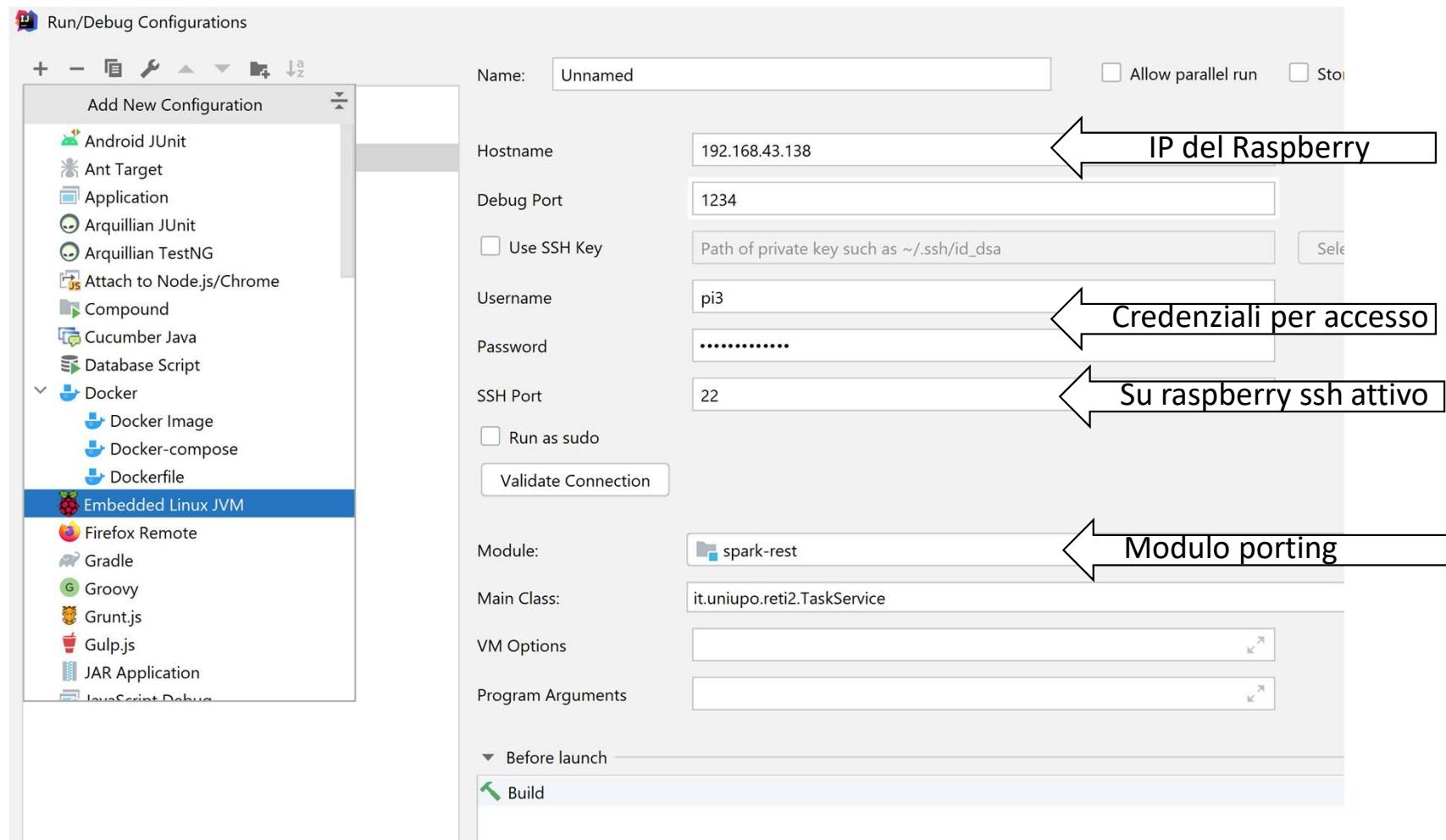
È possibile attivare:

- un servizio di comunicazione come mosquitto su Raspberry,
- oppure qualche componente dell'applicazione (es. Gestore IoT)

Porting dei componenti sviluppati su Raspberry Pi: installare il plugin



Definire una configurazione per Embedded Linux JVM



Eseguendo la configurazione ...

- Viene aperta una sessione ssh con il RaspberryPi
 - Viene trasferito il modulo in IdeaProjects
 - Viene avviato il programma sul Raspberry
-
- Esempio: spark-rest potete collegarvi a
`<IP RASPBERRY>:4567/ api/v1.0/ tasks`
per vedere i task memorizzati, oppure provare a eseguire un post con curl
per sperimentare l'aggiunta di un task:
`curl -X POST http://<IP RASPBERRY>:4567/api/v1.0/tasks -H 'Content-Type: application/json' -d '{"description":"Prova compito 1","urgent":0}'`

NOTA

Nel trasferimento del modulo il file task.db viene trasferito a non posizionato dove il programma se lo aspetta in src/main/resources quindi occorre spostarlo a mano oppure modificare il pathname nel sorgente

```
// On Raspberry - da sistemare
static private final String dbLoc = "jdbc:sqlite:src/main/resources/tasks.db";
// IL SEGUENTE PATH FUNZIONA PER IL DEPLOY SU RASPBERRY
//static private final String dbLoc = "jdbc:sqlite:classes/tasks.db";
```