Università di Padova

sweasabi@gmail.com

Corso di Ingegneria del Software A.A.:2022/2023

PoC

Versione documento: *V0.0.1*

Uso	Esterno
Destinatario	Committente Cliente

Indice

Capitolo 1

Tecnologie

- Angular
- Java Spring
- mqtt
- Java

Capitolo 2

Scopo

Lo scopo del POC è quello di dimostrare padronanza di una limitata selezione di tecnologie rilevanti al progetto. In particolare, le principali tecnologie utilizzate sono state Angular per la pagina front-end, Java Spring per l'api REST e Mosquitto come broker mqtt.

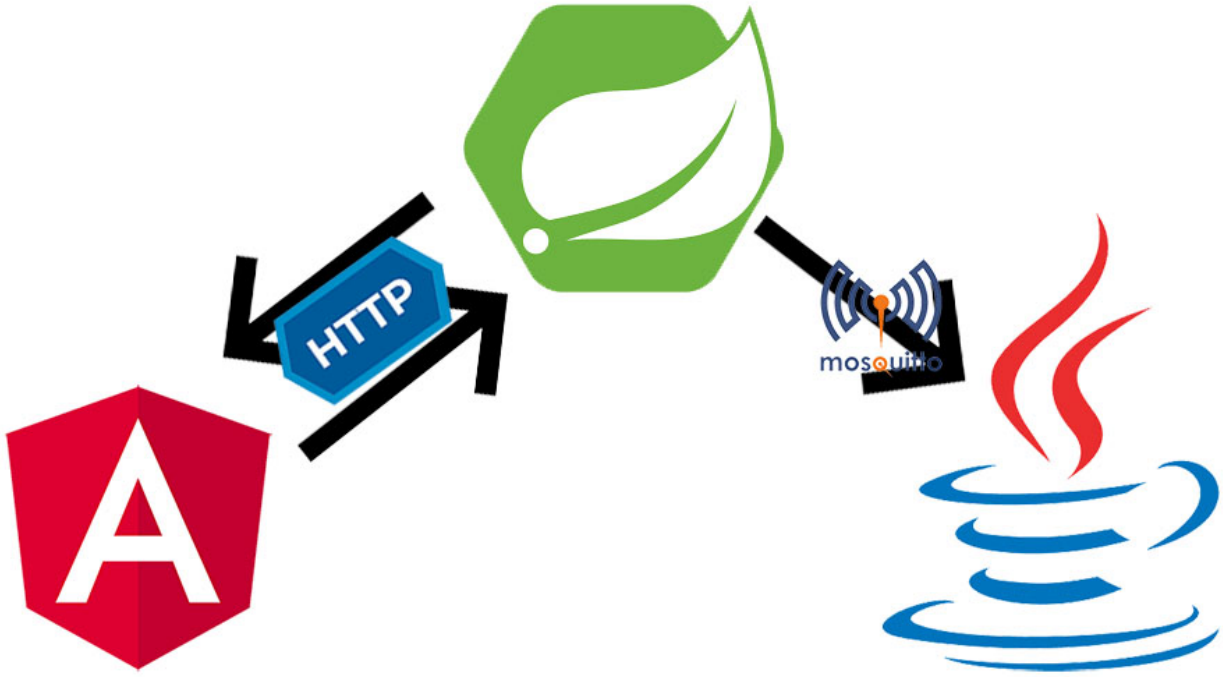

Figura 2.1: Simple visual representation

Capitolo 3

API Java Spring

3.1 Introduzione

Lo scopo dell'API è quello di fornire servizi REST alla pagina angular e comunicare con le applicazioni che vanno a svolgere la logica effettiva del progetto. In particolare, vengono offerte due GET (rispettivamente disponibili su Lamps e Lamps/id) e una PUT (anch'essa su Lamps/id). Nel progetto sarà presente un'ulteriore separazione tra API e servizi in modo da non esporli all'esterno (con l'API che si occuperà di verificare che la richiesta avvenga da un utente autorizzato e dirigendola verso la corretta applicazione), tuttavia essendo questo PoC puramente a scopo dimostrativo questa feature non è qui presente in quanto l'enfasi è stata posta sulla corretta comunicazione tra le diverse parti del progetto, ritenendo il processo di autenticazione vero e proprio superfluo in questo PoC (come confermato dal committente).

3.2 Funzionamento

3.2.1 Database

Per migliorare il funzionamento del PoC, è stato dotato di un piccolo database temporaneo a cui viene fatto un preload di alcune variabili durante l'inizializzazione del programma.

3.2.2 Richieste GET

Il progetto offre due GET, una su lamps e una su lamps/id. Queste GET ritornano rispettivamente i dettagli di tutte le lampade presenti (ad esempio il loro id, dove sono situate, o il loro status, ovvero se si tratta di una lampadina accesa o spenta) o quelli di una singola lampada corrispondente all'id selezionato.

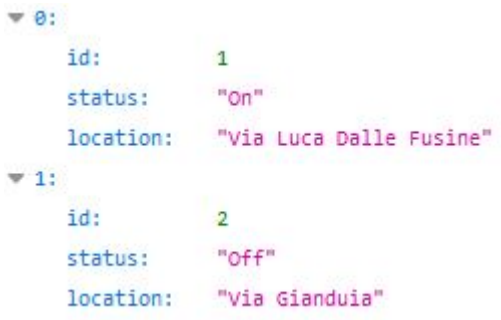

Figura 3.1: Api risponde a richiesta su /Lamps

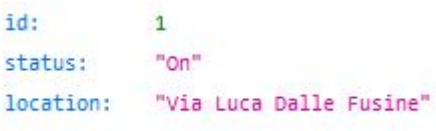

Figura 3.2: Api risponde a richiesta su /Lamps/1

3.2.3 Richiesta PUT

Come menzionato sopra il progetto offre anche una PUT, anch'essa a Lamps/id. Questo servizio consente di modificare i dettagli della lampadina corrispondente, in particolare di modificare il proprio stato (quindi accenderla o spegnerla).

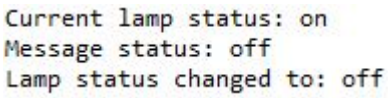

Figura 3.3: Richiesta PUT: La lampadina non è nello stato richiesto

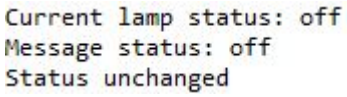

Figura 3.4: Richiesta PUT: La lampadina è già nello stato richiesto

3.2.4 Comunicazione MQTT

A ogni richiesta effettuata, l'API posta messaggi sui topic relativi alle lampadine utilizzando mqtt (nel caso del nostro progetto con broker mosquitto), messaggi che vengono successivamente letti dalle nostre applicazioni in back-end. Come spiegato sopra, questo passaggio è essenziale in quanto serve a comunicare con i servizi veri e propri facendogli eseguire le diverse operazioni richieste, tuttavia è più a scopo dimostrativo sul funzionamento della comunicazioni in questo PoC.

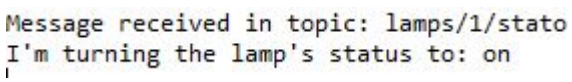

Figura 3.5: Richiesta PUT: Client legge il messaggio mqtt in arrivo dall'API

Capitolo 4

Angular

4.1 Introduzione

La pagina Angular mostra due differenti bottoni, uno di Fetch dello stato della API e un altro che modifica lo stato dell'entità[1], in questo bottone di modifica, il comportamento é quello di fetchare lo stato della lampada,per poi modificarlo nel suo complementare. Vengono Dichiarati i componenti Angular "app" e "lamp-button", che inglobano rispettivamente i fogli di stile e Markup, oltre che i rispettivi file typescript "component.ts" e ".module.ts"[2]. Vengono dichiarati i moduli "BrowserModule", essenziale per lanciare una app browser, e "HttpClientModule", essenziale per fare richieste HTTP, è così possibile implementare una GET, POST, DELETE oppure PUT. Nel Dominio del PoC si offrono tramite il metodo toggleLamp(), l'implementazione di una PUT per cambiare lo stato della lampada, innestato in una GET, capace di richiedere lo stato corrente della lampada stessa.

4.1.1 getData

Si offre il metodo getData, il quale da una chiamata GET all'url dello stato di tutte le lampade, quindi il loro id, il loro stato attuale[3] e la loro locazione.

4.1.2 toggleLamp

Si offre il metodo toggleLamp(), tale metodo fa una chiamata "this.http.get<any>" su lamps/1, si aspetta una risposta di tipo JSON e porta tale risposta alla richiesta innestata "this.http.put<any>", la quale verifica lo stato del sotto campo dell'oggetto Response, Response.status, e lo cambia con il suo complementare[4].

Ritorna, alla fine della PUT, un Console.log con l'oggetto Response, che in questo momento temporale e' pari al risultato della operazione[5].

[1] nel dominio del POC, la prima entitá ID=1
[2] per il componente app
[3] acceso = on, spento = off
[4] status: Response.status == 'on' ? 'off' : 'on'
[5] e' stato fatto l'update, se modifica portata a buon fine

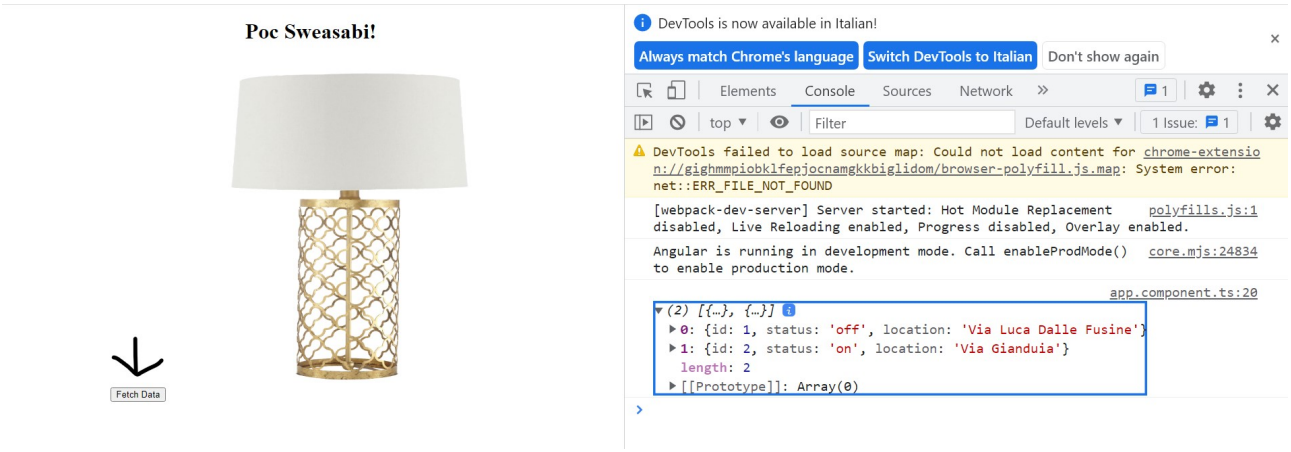

Figura 4.1: Button fa richiesta su /Lamps

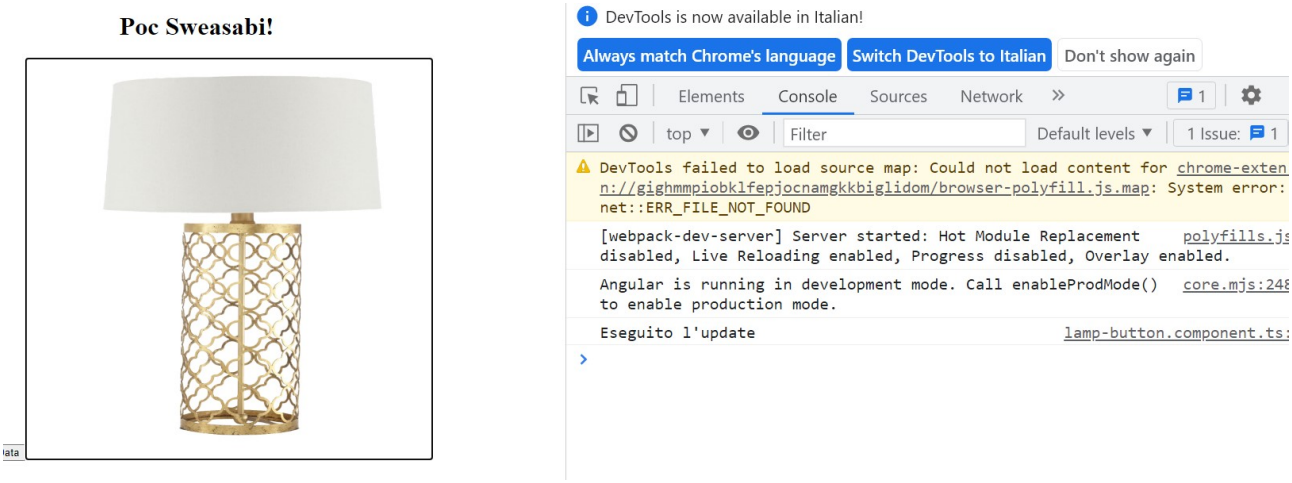

Figura 4.2: Button fa cambio di stato su /Lamps/1

```
Lamp 1has been requested
Lamp status: on
Message status: off
```

Figura 4.3: Button fa cambio di stato su /Lamps/1