

Sviluppo di una Applicazione Internet-of-Things basata su IoT-Data-Simulator

Tesi in Tecnologie Web T

Relatore:

Prof. Paolo Bellavista

Presentata da:

Michele Proietti

Matricola 0000882929

Sommario

L'utilizzo di oggetti smart collegati alla rete si sta diffondendo sempre di più portando ad una sostituzione ed utilizzo sempre più frequente di quest'ultimi a discapito di quelli tradizionali. Un esempio potrebbe essere quello dell'utilizzo di lampadine smart che, tramite l'utilizzo di sensori, riescono a rilevare il livello di luminosità di una singola stanza o di un intero appartamento e di conseguenza decidere autonomamente quante e quali lampade accendere o spegnere. L'obiettivo di questo elaborato è, in prima battuta, fornire una panoramica generale sull'argomento dell'Internet Of Things, per poi descrivere il funzionamento di un piccolo progetto basato sull'utilizzo di un simulatore IOT.

Indice

Sommario	I
Introduzione	1
1 Internet Of Things	3
1.1 La Definizione	3
1.2 La storia	5
1.3 Trend di Mercato	8
1.4 Applicazioni IoT e Settori applicativi	11
1.4.1 Applicazioni IoT	11
Applicazioni Iot Consolidate	11
Applicazioni Iot Sperimentali	12
Applicazioni Iot Embrionali	12
1.4.2 Settori Applicativi	13
Smart City	14
Smart Home	14
Smart Building	15
Smart Mobility	15
Smart Health	15
Smart Agriculture	15
1.5 Architettura IoT	16
1.5.1 Lo Stack Architetturale	16
Perception Layer	17
Network Layer	18
Middleware Layer	18
Application Layer	18

Business Layer	19
2 IoT-data-simulator	20
2.1 Perché utilizzare un simulatore?	20
2.2 Concetti fondamentali	22
2.3 L'interfaccia grafica	24
2.3.1 Homepage	24
2.3.2 Session	25
2.3.3 Data Definitions	26
2.3.4 Device	28
Device Injection	28
2.3.5 Target System	30
2.4 Generazione dei dati	32
3 Il progetto	33
3.1 L'idea	33
3.2 Il funzionamento	35
3.3 Il Simulatore	36
3.3.1 Abitazione: Data Definition	37
3.3.2 Abitazione: Regole temporali	38
3.3.3 Abitazione: Dispositivi	39
3.3.4 Abitazione: Regole di iniezione dei dispositivi . .	39
3.3.5 Abitazione: Regole di generazione dei dati	40
Generazione dei valori associati al livello di lumi- nosità	41
3.3.6 Abitazione: Sistemi di destinazione	43
3.4 La raccolta dei dati	45
3.4.1 Formato dei dati	45
3.4.2 Breve introduzione al protocollo MQTT	45
3.5 Implementazione del protocollo MQTT	47
3.6 API	51
3.6.1 Creazione del progetto	51
3.6.2 Implementazione delle API	51
3.7 L'applicazione	54
3.7.1 Fase 1: Caricamento della configurazione	54

3.7.2	Fase 2: Acquisizione delle rilevazioni dei sensori .	55
3.7.3	Fase 3: Caricamento dello stato delle luci	57
3.7.4	Fase 4: Elaborazione dei dati e stampa dei risultati	57
	Fase 4.1: Controllo dello stato delle luci	59
	Fase 4.2: Modifica dello stato delle luci	61
	Fase 4.3: Stampa dei risultati	62
3.8	Test del Progetto	63
3.8.1	Test subscriber.js	63
3.8.2	Test SmartLight	63
Conclusioni		65
Bibliografia e Sitografia		67
Ringraziamenti		69

Introduzione

Anche se non ce ne rendiamo conto stiamo vivendo e assistendo ad una rivoluzione della rete che ha portato e porterà numerosi cambiamenti. Ormai abbiamo le capacità tecnologiche di rendere le "cose" intelligenti permettendo a queste di poter agire senza necessitare dell'intervento umano.

Infatti tali oggetti sono in grado di rendersi riconoscibili e acquisiscono intelligenza potendo inviare informazioni ad altri oggetti e reperire informazioni fornite da altri. Essi sono in grado di dialogare e interagire tra loro attraverso sensori senza l'intervento umano mediante l'utilizzo di protocolli specifici e interconnessioni.

Questo ha portato ad un utilizzo sempre più frequente di questi oggetti smart a discapito di quelli tradizionali, creando dei veri e propri ecosistemi in cui tali oggetti vivono e interagiscono tra di loro al fine di raggiungere un particolare obiettivo dipendente dall'ambito in cui sono utilizzati.

L'elaborato si pone l'obiettivo di fornire una visione generale sull'argomento dell'*Internet Of Things* per poi mettere in pratica i concetti teorici acquisiti presentando un piccolo progetto che sfrutta le funzionalità messe a disposizione dal simulatore **IoT-data-simulator** fornito dall'*IBA Group*.

In particolare il progetto simulerà il caso di un abitazione intelligente in cui, per ogni stanza, è stato installato un sensore di luminosità che periodicamente invierà i dati raccolti. Tali dati verranno poi processati al fine di decidere se cambiare lo stato delle luci (acceso → spento, spento → acceso) per mantenere un'illuminazione ottimale all'interno di ogni

stanza.

L'elaborato si compone in 3 capitoli. Il primo capitolo si occuperà di introdurre l'*Internet Of Things* facendo una panoramica generale sull'argomento spiegando cos'è e come questo si è affermato nel corso degli anni.

Il secondo capitolo si occuperà di descrivere in modo dettagliato il simulatore che è stato utilizzato per sviluppare il progetto, andandone a dettagliare tutte le parti e il funzionamento generale.

Il terzo e ultimo capitolo sarà dedicato alla descrizione del progetto in essere e si occuperà di descrivere le diverse tecnologie utilizzate e le diverse scelte implementative che sono state fatte.

Capitolo 1

Internet Of Things

Questo primo capitolo si pone l'obiettivo di dare una visione generale riguardo l'argomento dell'*Internet Of Things*, andando per prima cosa a definire una cronistoria riguardo l'argomento. In secondo luogo, verranno riportati i principali trend che riguardano l'argomento, con un piccolo sguardo a sviluppi futuri. Nella terza sezione verranno presentati i principali settori e ambiti in cui è ed è possibile utilizzare i concetti su cui si fonda l'Iot. Infine nell'ultima sezione, un po' più tecnica, verrà descritta l'architettura dell'IoT cercando comunque di non concentrarsi su un'unica tecnologia ma rimanendo il più possibile nel generale.

1.1 La Definizione

Internet of Things - letteralmente “Internet delle cose” - è l'espressione utilizzata per definire un qualsiasi sistema di dispositivi fisici che ricevono e trasferiscono i dati e informazioni sfruttando una rete wireless a cui sono tutti collegati.

Ogni dispositivo è dotato di una componente software installata che permette al singolo dispositivo di rendersi riconoscibile agli altri dispositivi, comunicare con gli altri sfruttando particolari protocolli ed elaborare i dati che vengono raccolti. L'obiettivo di questo ecosistema di dispositivi generalmente eterogenei è quello di comunicare tra di loro, tramite l'invio di informazione, e portare avanti un certo compito senza, o comunque in misura molto piccola, l'intervento umano.

Quindi, almeno in prima battuta, un oggetto deve presentare due fondamentali caratteristiche:

- Deve rendersi riconoscibile agli altri oggetti all'interno del complesso in cui "vive".
- Deve poter interagire con gli altri dispositivi senza necessitare dell'intervento umano per funzionare.

Questi concetti possono essere applicati in numerosi ambiti, come ad esempio:

- **La Domotica.** Il concetto delle Smart Home sta diventando sempre più presente nella nostra quotidianità. Si pensi alle migliaia di aziende che forniscono dei servizi di domotica con i quali ad esempio è possibile installare dei sistemi di rilevazione della temperatura in grado di gestire l'accensione e lo spegnimento del sistema di riscaldamento in modo completamente autonomo.
- **Smart Building.** Interi edifici in grado di gestire in modo completamente autonomo l'utilizzo delle risorse riducendo lo spreco delle stesse.
- **Smart Mobility.** Sempre più presente nelle grandi città italiane e all'estero. Un esempio è sicuramente *Mobike* che offre un servizio di bike sharing a flusso libero senza l'uso di stalli per il parcheggio.

Questi sono semplici e pochi esempi che però ci danno un'idea di come il fenomeno dell'*Internet Of Things* stia prendendo sempre più piede nella nostra quotidianità.



Figura 1.1: IoT examples

1.2 La storia

"Quando la telefonia senza fili sarà perfettamente applicata, la Terra si trasformerà in un enorme cervello, quale di fatto è, e tutte le cose saranno parte di un intero reale e pulsante.". Questo è un estratto dell'intervista rilasciata dall'inventore serbo Nikola Tesla nel 1926 al reporter statunitense John B. Kennedy, per il magazine *Colliers*[3]. Già nel 1926, durante questa intervista, Nikola Tesla aveva già intuito che, con il passare del tempo, lo sviluppo tecnologico avrebbe portato ad un mondo in cui tutte le cose, anche le più semplici, sarebbero state connesse tra di loro e quindi in grado di comunicare. Ovviamente era solo il 1926, e quella di Tesla era di fatto solo un'intuizione.

Infatti per sentir parlare veramente di Internet Of Things dobbiamo ancora ripercorrere numerose tappe.

Nel 1989 Tim Berners-Lee al CERN di Ginevra propone il concetto del *World Wide Web* così come ancora oggi lo conosciamo.

Nel 1990, durante la fiera annuale organizzata dall'UBM, John Romkey presenta quello che per molti è considerato come il primo dispositivo IoT. Romkey presentò un tostapane connesso alla rete tramite protocol-

lo TCP/IP. Per accendere e spegnere il tostapane veniva utilizzato un protocollo di rete detto *SNMP* (*Simple Network Management Protocol*) che consente di semplificare la configurazione, gestione e supervisione di apparati collegati in una rete.

Finalmente arriviamo al 1999 quando il ricercatore del MIT, Kevin Ashton, conìò il neologismo “Internet of Things” durante una presentazione per definire un network globale e dinamico di dispositivi dotati di indirizzo IP, che si scambiano dati senza l’intervento umano. Quando Ashton usò per la prima volta questa espressione, stava lavorando all’ottimizzazione della *supply chain*, presso la *Procter & Gamble* con lo scopo di attirare l’attenzione del top management sui tag RFID. Nonostante Ashton riuscì nel suo intento, il neologismo *Internet of Things* inizialmente non si diffuse, entrando nel vocabolario quotidiano solo una decina di anni più tardi, a tal punto da diventare l’espressione più popolare per descrivere il nuovo mondo interconnesso.

Nel 2008 viene organizzata il primo congresso sull’IoT, a Zurigo.

Nel 2010 il premier cinese Wen Jiabao, a fronte della crisi finanziaria e di altre questioni sensibili del Paese, sceglie l’IoT come settore chiave in cui investire. Nello stesso anno viene creata la start up Nest, successivamente acquistata da Google nel 2014, che inizia la produzione di elettrodomestici intelligenti. In particolare il primo prodotto è stato il *Nest Learning Thermostat*, ovvero un termostato intelligente in grado di apprendere le abitudini degli utenti e, di conseguenza, ottimizzare il programma di riscaldamento. Questo è stato il primo termostato IoT al mondo alimentato con machine learning.

Nel 2011 viene rilasciato pubblicamente il nuovo protocollo Internet Ipv6, in successione all’Ipv4, che amplia i servizi offerti dalla rete e semplifica la gestione degli indirizzi IP.

Quindi anche solo ricordando questi pochi momenti nel corso della storia si capisce come il concetto dell’IoT sia notevolmente mutato nel corso del tempo e come sia diventato un qualcosa sempre più presente nella quotidianità. Infatti analizzando i dati raccolti dal 2010 ad oggi, tramite Google Trends[15]:



Figura 1.2: Google Trends

si osserva come l'interesse per l'argomento dell'IoT sia cresciuto in modo esponenziale.

1.3 Trend di Mercato

Il 29 ottobre 2020 in qualità di principale società di ricerche di mercato sulla *trasformazione digitale (DX)* al mondo, l'*International Data Corporation (IDC)* ha presentato un articolo dal titolo: *IDC FutureScape: Worldwide Digital Transformation 2021 Predictions* [5].

Secondo l'IDC nonostante una pandemia globale, gli investimenti per la trasformazione digitale (DX) stanno ancora crescendo a un tasso di crescita annuale composto (CAGR) del 15,5% dal 2020 al 2023 e si prevede che si avvicineranno a 6,8 trilioni di dollari entro il 2023.

In particolare le previsioni che sono emerse dall'articolo sono le seguenti:

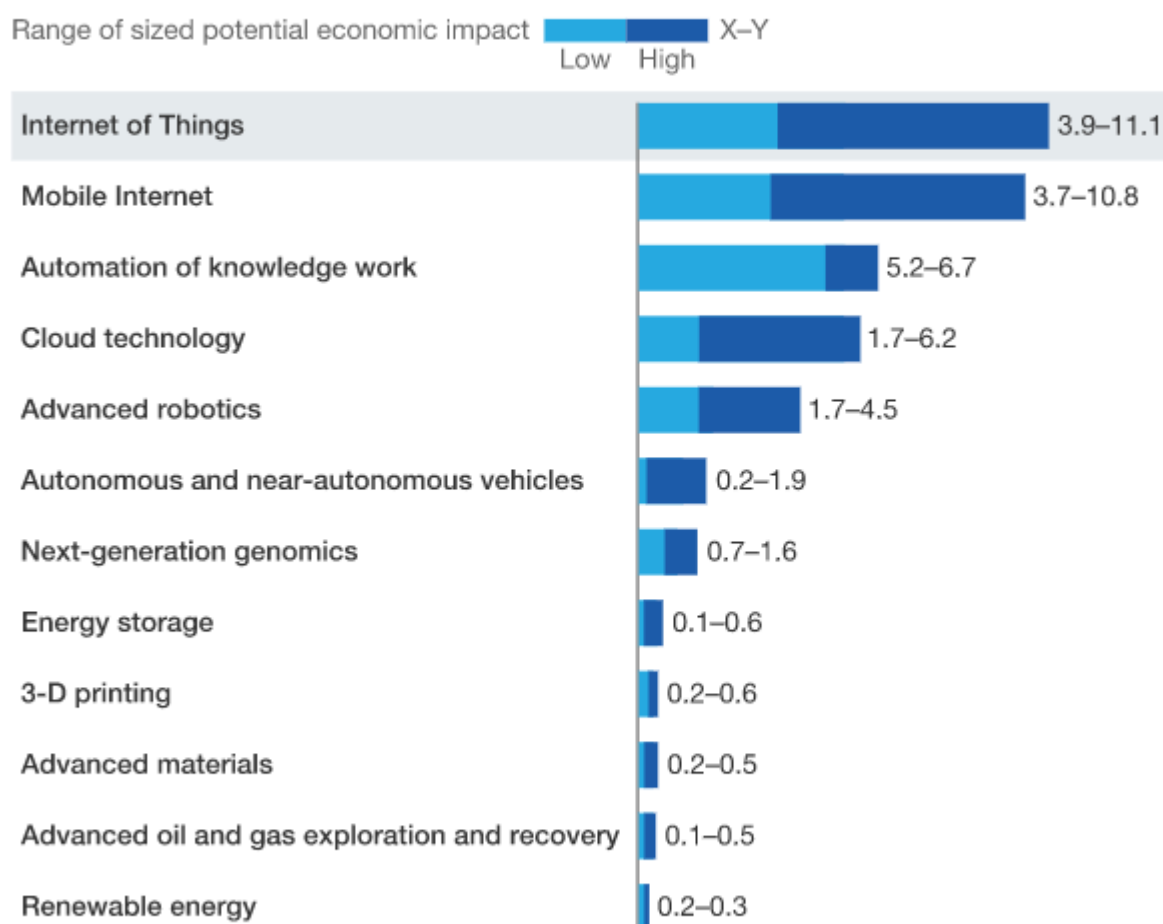
- **Prediction 1: Accelerated DX Investments Create Economic Gravity.** Secondo cui entro il 2022 il 65% del PIL globale sarà utilizzato per la digitalizzazione.
- **Prediction 2: Digital Organization Structures and Roadmaps Mature.** Secondo cui il 75% delle organizzazioni disporrà di roadmap complete per l'implementazione della trasformazione digitale (DX), in aumento rispetto al 27% attuale, entro il 2023.
- **Prediction 3: Digital Management Systems Mature.** Secondo cui, entro sempre il 2023, il 60% dei leader nelle organizzazioni del G2000 avrà cambiato il proprio orientamento gestionale dai processi ai risultati stabilendo dei modelli operati più agili, innovativi e empatici.
- **Prediction 4: The Rise of the Digital Platform and Extended Ecosystems.** Secondo cui entro il 2025, a causa della volatilità delle condizioni globali, il 75% dei leader aziendali sfrutterà le piattaforme digitali e le capacità degli ecosistemi per adattare le loro "value chains" per nuovi mercati, industrie ed ecosistemi.
- **Prediction 5: A Digital First Approach.** Secondo cui già nel 2021 il 60% delle imprese investirà pesantemente nella digitalizzazione dell'esperienza dei dipendenti, trasformando il rapporto tra datori di lavoro e dipendenti.

- **Prediction 6: Business Model Reinvention.** Secondo cui già nel 2021, almeno il 30% delle organizzazioni accelererà l'innovazione al fine di rendere le proprie attività a prova di futuro.
- **Prediction 7: Sustainability and DX.** Entro il 2022 la maggior parte delle aziende investirà nella possibilità di rendere ancor più stretto il connubio tra digitalizzazione e sostenibilità.
- **Prediction 8: Digitally Native Cultures.** Entro il 2025 il 50% delle imprese investirà su delle organizzazione basate sul cliente ma comunque guidate dai dati.
- **Prediction 9: Accelerating Digital Experiences.** Entro il 2022 il 70% delle aziende accelererà riguardo all'utilizzo di tecnologie digitali, trasformando i processi aziendali esistenti per favorire il coinvolgimento dei clienti, la produttività dei dipendenti e la resilienza aziendale.
- **Prediction 10: Business Innovation Platforms.** Entro il 2023, il 60% delle aziende del G2000 costruirà la propria piattaforma di innovazione aziendale per supportare l'innovazione e la crescita nella nuova normalità.

Oltre alle previsioni dell'IDC si riporta anche lo studio eseguito dalla *McKinsey & Company* [1] secondo cui entro il 2025 l'IoT avrà un potenziale impatto economico totale di ben 11,1 trilioni di dollari all'anno. In effetti, l'IoT sarà la più grande fonte di valore di tutte le tecnologie dirompenti, prima ad esempio di Internet mobile, cloud computing e robotica avanzata.

L'articolo pubblicato fa emergere come l' *Internet Of Things* sarà in prima posizione rispetto ad altri settori di ricerca:

The Internet of Things will have substantial economic impact by 2025 among a list of disruptive technologies.



McKinsey&Company | Source: McKinsey Global Institute analysis

Figura 1.3: McKinsey Global Institute analysis

1.4 Applicazioni IoT e Settori applicativi

Parlare delle possibili applicazioni del paradigma dell'*Internet Of Things* risulta essere molto complesso. Infatti siamo arrivati ad un punto in cui la maggior parte dei dispositivi che ci circondano sono in grado di connettersi ad una rete internet e quindi, potenzialmente, poter comunicare con altri dispositivi.

Per questo motivo descrivere i diversi ambiti applicativi non risulta essere così semplice ma comunque in questa trattazione si cercherà di schematizzarli il più possibile distinguendo tre categorie principali di applicazioni dedicando ad ognuna una sezione, tale suddivisione è fatta sulla base della **maturità delle applicazioni** in essere.

Inoltre, sempre per maggior chiarezza, si riporta anche una suddivisione sulla base dei settori in cui sono e possono essere sfruttati i concetti dell'*Internet Of Things*.

1.4.1 Applicazioni IoT

Applicazioni Iot Consolidate

La prima categoria che andiamo a trattare riguarda quelle applicazioni che vengono identificate come **consolidate**.

Per consolidate si considerano tutte quelle applicazioni che sono direttamente e facilmente realizzabili in virtù delle capacità tecnologiche attuali. Quindi in questa prima categoria rientrano tutte quelle applicazioni che o sono già state realizzate oppure sono facilmente realizzabili.

Qualche esempio?

Possibili applicazioni che rientrano in questa categoria sono:

- Sistemi per la videosorveglianza all'interno delle abitazioni o a livello industriale.
- Sistemi creati ed utilizzati per il tracciamento degli oggetti di valore.
- Sistemi per il monitoraggio del traffico all'interno delle città, che potrebbero essere installati per scopi statistici o gestionali.

- Sistemi per la gestione degli elettrodomestici come lavatrici, frigoriferi..ecc

Applicazioni Iot Sperimentali

La seconda categoria che andiamo a trattare riguarda quelle applicazioni che vengono identificate come **sperimentali**.

Con l'espressione **sperimentali** si intendono tutte quelle applicazioni che cercano di seguire a pieno il paradigma dell'*Internet Of Things*, cercando quindi di creare ecosistemi di oggetti intelligenti sempre più complessi ed estesi.

Qualche esempio?

- L'**eHealth**, ovvero l'utilizzo del paradigma dell'*Internet Of Things* nel settore sanitario, in cui potrebbero essere utilizzati dei sistemi di monitoraggio dei pazienti per ridurre il costo dei servizi ospedalieri ed aumentare l'efficienza di quest'ultimi.
- Soluzioni basate su tecnologie **RFId** per la **supply chain** utilizzate all'interno di grandi aziende

Applicazioni Iot Embrionali

L'ultimo grado è quello delle cosiddette applicazioni **embrionali**.

Per **applicazioni embrionali** si intendono tutte quelle applicazioni che, attualmente, non sono completamente realizzabili. Infatti rientrano in questa categoria tutte quelle applicazioni che guardano al futuro e attualmente sono ancora nelle prime fasi sperimentali.

Per quanto riguarda quest'ultima categoria le applicazioni che si trovano in una fase sperimentale abbastanza avanzata rientrano nel settore delle **Smart Grid**, ovvero quel settore legato alla gestione del consumo elettrico e dell'energia in generale.

1.4.2 Settori Applicativi

Abbiamo già ribadito il fatto che i concetti su cui si basa l'*Internet Of Things* presentano una forte versatilità, nel senso che possono essere applicati ed adattati a numerosi ambiti. Per questo motivo in questa sezione si è scelto di presentare i principali settori in cui il paradigma dell'IoT viene applicato.



Figura 1.4: Settori Applicativi

Smart City

Le città intelligenti utilizzano delle strategie di pianificazione urbanistica in grado di migliorare la qualità della vita all'interno delle città. L'obiettivo principale è quello di soddisfare al meglio le esigenze dei cittadini.

Tutto questo è possibile sfruttando delle tecnologie che permettono di relazionare le infrastrutture con gli abitanti della città. Ad esempio dei sistemi innovativi per la gestione e smaltimento dei rifiuti oppure dei sistemi per il controllo del traffico.

Smart Home

Il concetto delle Smart Home - casa intelligente - rappresenta tutto quel settore che si pone l'obiettivo di fornire dei sistemi che possono essere utilizzati per migliorare la vita all'interno dell'ambiente domestico. Possibili esempi possono essere regolare la temperatura della casa a distanza, oppure sensori di rilevamento del numero di persone all'interno dell'abitazione. Inoltre in questo settore rientra anche tutto ciò che concerne la **Domotica**, ovvero tutto l'insieme di sistemi e dispositivi che permettono di migliorare il comfort e l'efficienza della casa attraverso funzionalità integrate e cablate nell'impianto elettrico.

Smart Building

Il concetto di Smart Building - edifici intelligenti - anche se a prima vista può sembrare molto simile a quello di Smart Home, in realtà è molto diverso.

Infatti, a differenza della Smart Home, con gli Smart Building ci si pone l'obiettivo di realizzare dei sistemi in grado di gestire interi palazzi ed uffici. Quindi dotare quest'ultimi di oggetti intelligenti in grado di interagire con l'ambiente interno, ad esempio per quanto concerne la gestione della luce e dell'energia elettrica.

Smart Mobility

Il concetto dello Smart Mobility è strettamente legato e per certi aspetti necessario per definire il concetto di Smart City.

Infatti il concetto della mobilità intelligente sta prendendo sempre più piede all'interno delle città, spaziando dal semplice Car e Bike Sharing fino a progetti molto più grandi ad esempio legati alla realizzazione di treni controllati da applicazioni IoT.

Smart Health

Questo termine viene utilizzato per indicare tutte quelle applicazioni utilizzate in ambito sanitario. Lo scopo principale di questi sistemi è quello di rendere più efficienti i servizi sanitari, come ad esempio i servizi ospedalieri.

In questo settore ad esempio rientrano quei sistemi per il monitoraggio delle condizioni dei pazienti oppure per la gestione delle emergenze.

Smart Agriculture

Conosciuto anche come **Agrifood**, con il termine Smart Agriculture si fa riferimento a tutto quell'insieme di servizi che potrebbero essere implementati al fine di sfruttarli in campo ambientale.

Possibili esempio potrebbero essere dei sistemi basati su sensori utilizzati per il rilevamento delle condizioni meteorologiche e il monitoraggio delle

loro variazioni; oppure dei sistemi per il monitoraggio della crescita dei raccolti.

1.5 Architettura IoT

L'obiettivo di questo primo capitolo è quello di presentare in generale il concetto dell'*Internet Of Things*.

Quindi in questa ultima sezione si è scelto di presentare, nel modo più generale possibile quindi senza concentrarsi su una particolare implementazione, lo standard utilizzato in termini di *Stack Architetturale*.

1.5.1 Lo Stack Architetturale

Come abbiamo già ribadito il paradigma dell'*Internet Of Things* può essere applicato a numerosi ambiti. Per questo motivo, in virtù del tipo di applicazione che si vuole progettare e del tipo di interazione che si vuole avere (tra dispositivi e dispositivi o tra dispositivi e utenti), ogni progetto IoT risulta essere differente da tutti gli altri.

Anche se ogni progetto e applicazione IoT risulta essere diversa da tutte le precedenti e da quelle che verranno è comunque possibile parlare di uno standard in termini di **Stack Architetturale**.

In accordo con il *Conference Paper* del Dicembre 2012 dal titolo *Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges* [8], pubblicato su *Research Gate*; lo standard architetturale prevede una divisione dello stack in 5 livelli fondamentali, mostrati nella figura sottostante

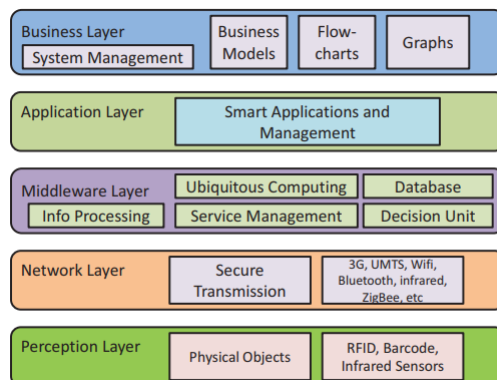


Figura 1.5: Stack Architetturale Iot

Ovviamente questi livelli non sono indipendenti anzi fortemente correlati tra di loro, in quanto per ogni livello le informazioni raccolte od elaborate vengono passate al livello superiore per essere processate. Segue una descrizione livello per livello.

Perception Layer

Conosciuto anche come **Device Layer** rappresenta il livello più basso dello stack.

Con questo layer si fa riferimento alla componente fisica dell'architettura quindi agli oggetti fisici e ai sensori, quest'ultimi possono essere implementati secondo diverse tecnologie ma tra le più comuni ricordiamo RFID, codici a barre 2D, sensori a infrarossi..ecc

Questo primo livello risulta essere fondamentale in quanto rappresenta la base fisica su cui fanno riferimento tutti i livelli superiori. Infatti questo livello si occupa fundamentalmente dell'identificazione e raccolta di informazioni specifiche sugli oggetti da parte dei sensori. Ovviamente tali informazioni variano in base al tipo di applicazione che si vuole progettare, alcuni esempi possono essere informazioni riguardo la temperatura, l'accelerazione, la posizione..ecc

Network Layer

Il *Network Layer* o *Transmission Layer* si colloca in seconda posizione appena sopra il *Device Layer*.

Come suggerisce il nome la funzione che svolge questo livello è una funzione di trasmissione dati dal livello sottostante al *Middlware Layer*.

Infatti una volta ottenute le informazioni dal livello sottostante, il *Network Layer* ha il compito di trasferire **in modo sicuro** le informazioni al livello superiore. Anche questo presenta diverse implementazioni in base alla tipologia di trasmissione utilizzata che però può essere divisa in due tipologie fondamentali: **Cablata** o **Wireless**.

Altra differenziazione può essere fatta in base alla tecnologia che viene utilizzata per la trasmissione, che ovviamente dipende dal tipo di sensori che si è scelto di utilizzare nel livello sottostante; tra le più comuni il 3G, Wifi, Bluetooth..ecc

Middlware Layer

Il terzo livello dello stack è il *Middlware Layer*. Questo ha un ruolo cruciale all'interno dello stack in quanto ha il compito di gestire le diverse informazioni che riceve dal *Network Layer*, a sua volta ottenute dal *Device Layer*, al fine di renderle disponibili al livello superiore. Per questo motivo in questo livello i dati ricevuti vengono processati ed infine memorizzati, ad esempio, con l'utilizzo di database, in modo tale da renderli disponibili al livello superiore.

Application Layer

Il quarto livello dello stack è l'*Application Layer*. Con questo livello iniziamo a descrivere la parte dello stack più vicina all'utente utilizzatore, quindi sempre più indipendente dalle scelte implementative che sono state fatte ai livelli inferiori.

In particolare questo livello ha il compito di gestire le diverse applicazioni IoT, in virtù dei dati che ottiene dal *Middlware Layer*. Possibili esempi di applicazioni implementabili sono le applicazioni Smart Health,

Smart Home, Smart City...ecc. Quindi in questa fase emerge la grande versatilità dell'utilizzo del paradigma dell'*Internet Of Things*.

Business Layer

Ultimo, ma non per importanza, è il *Business Layer*.

Questo ultimo livello svolge un ruolo gestionale all'interno dell'architettura, in quanto ha il compito di gestire tutto il sistema Iot sottostante in virtù dei dati ottenuti dall'*Application Layer* tramite l'utilizzo di grafici, diagrammi di flusso...ecc.

Quest'ultimo livello risulta essere fondamentale all'interno dell'architettura in quanto, in virtù dei risultati ottenuti, permette ad esempio di determinare più facilmente le attività da dover svolgere in futuro oppure scegliere con maggior accuratezza le strategie di business da seguire.

Capitolo 2

IoT-data-simulator

Questo secondo capitolo della trattazione si pone l'obiettivo di descrivere nel modo più completo possibile le caratteristiche e funzionalità fornite dal simulatore *IoT-data-simulator* messo a disposizione dall'*IBA-Group-IT* [4].

L'*IoT-data-simulator* è un progetto *open-source* sviluppato da Vadzim Kazak ed altri collaboratori, che fornisce un generico simulatore Iot tramite il quale è possibile simulare i dati dei dispositivi IoT con grande flessibilità. Infatti, tramite una semplice interfaccia grafica, permette di simulare anche architetture complesse composte da numerosi dispositivi. Il vantaggio più grande del simulatore risiede nella sua forte versatilità. Infatti, essendo un simulatore generico, non è legato ad una particolare tecnologia e quindi è possibile adattare ed utilizzare le sue funzionalità in virtù dell'utilizzo che se ne vuole fare.

2.1 Perché utilizzare un simulatore?

Ogni volta che si decide di progettare e sviluppare un'applicazione è buona norma testare quell'applicazione al fine di rilevare possibili bug o problemi prima di installarla sui dispositivi fisici per cui è stata implementata.

Le applicazioni e i progetti IoT ovviamente non fanno eccezione. Infatti per quanto l'utilizzo di sensori e dispositivi del mondo reale sia necessario per i test di integrazione finali per assicurarsi che il sistema funzioni

dall'inizio alla fine senza sorprese, è molto poco pratico utilizzare i dispositivi reali durante le fasi di sviluppo e test iniziali in cui i test tendono ad essere più rapidi, più brevi, fallendo più velocemente e più spesso.

Per questo motivo nelle prime fasi di implementazione potrebbe essere utile utilizzare un simulatore che permette, almeno nelle prime fasi, di facilitare il lavoro e ridurre i tempi di risoluzione dei problemi.

Alcuni vantaggi che possono emergere dall'utilizzo di un simulatore IoT sono:

- Evitare di mettere in esecuzione tutta l'architettura fisica semplicemente per effettuare un test.
- Poter generare i dati inviati dai dispositivi in termini di mole, tipo e frequenza. Questo permette al progettista di simulare il più possibile la realtà oppure creare scenari verosimili al fine di testare i limiti dell'applicazione implementata.
- Permette di testare puntualmente le diverse parti dell'applicazione senza dover mettere in esecuzione l'intera architettura fisica.

Quindi possiamo dire che poter utilizzare un simulatore all'interno di un progetto IoT rappresenta un vantaggio non indifferente. Infatti, oggi giorno, l'avvento dell'IoT ha reso la simulazione ancor più indispensabile per progettare i nuovi dispositivi e quindi sono nate sempre più aziende che cercano di lavorare su questo aspetto. Un esempio è l'azienda *ANSYS*[2], che non produce oggetti tangibili ma la sua missione da sempre è il supportare gli ingegneri a sviluppare i migliori prodotti possibili attraverso l'approccio *Simulation-Driven Product Development*. L'obiettivo di questo approccio è facilitare il lavoro dei progettisti e migliorare la qualità del progetto, un esempio di possibile tecnica è quella dei *digital twins* secondo cui ogni oggetto fisico ha una sua copia virtuale, un gemello digitale. I dati raccolti dall'oggetto fisico possono essere comparati con quelli della copia virtuale per identificare possibili problemi di performance e prevedere soluzioni preventive.

2.2 Concetti fondamentali

Prima di andar a descrivere dettagliatamente il simulatore, andando a descriverne anche l'interfaccia grafica, dobbiamo introdurre 4 concetti fondamentali che saranno necessari al fine di capire al meglio le funzionalità messe a disposizione dallo stesso.

Per descrivere al meglio il simulatore è necessario introdurre i concetti di:

- **Session.**

Questa rappresenta l'entità principale dell'applicazione composta. Infatti è possibile generare dei dati ed inviare questi, ad esempio a sistemi esterni, solo durante l'esecuzione di una sessione.

La sessione è composta da numerose entità, tra cui:

- **Data Definition** (opzionale)
- **Timer**, tramite il quale è possibile definire le regole temporali che verranno poi applicate alla generazione dei dati.
- **Device** (opzionale)
- **Processing rules**, che rappresentano l'insieme delle regole che vengono seguite per la generazione dei dati.
- **Target System**

- **Data Definition.**

Tramite il concetto di *Data Definition* è possibile descrivere la struttura dati che verrà poi inviata ad esempio ad un sistema di destinazione. Questa è composta da due sotto-entità: il **Dataset** e lo **Schema**, i quali non sono mutuamente esclusivi tra di loro in quanto possono contemporaneamente far parte dello stesso *Data Definition*.

- **Dataset.**

Necessario se non è fornito uno Schema. Questo rappresenta un file *.json* o *.csv* che ad esempio può essere caricato

dall'utente tramite l'interfaccia grafica. Se si decide di caricare un Dataset il suo contenuto verrà riprodotto dal simulatore durante l'esecuzione della sessione associata.

– **Schema.**

Necessario se non è fornito un Dataset. Tramite lo *Schema* è possibile descrivere la struttura del set di dati o i dati che verranno generati in fase di esecuzione. Lo schema può essere derivato dal set di dati o creato da zero tramite il costruttore dell'interfaccia utente. Se lo Schema non viene specificato i dati verranno generati sulla base della funzione JavaScript personalizzata fornita dall'utente.

• **Target System.**

Come suggerisce il nome, questo permette di definire un sistema esterno a cui verranno inviati i dati generati.

• **Device.**

Fornisce un modo per simulare un dispositivo IoT, in virtù delle sue caratteristiche e delle informazioni che raccoglie.

2.3 L'interfaccia grafica

2.3.1 Homepage

Dopo aver messo in esecuzione il simulatore tramite *Docker*, quest'ultimo sarà disponibile su **localhost** sulla porta **8090**. Infatti accedendo, tramite browser, all'url *localhost:8090* viene mostrata l'interfaccia grafica del simulatore che si presenta in questo modo:

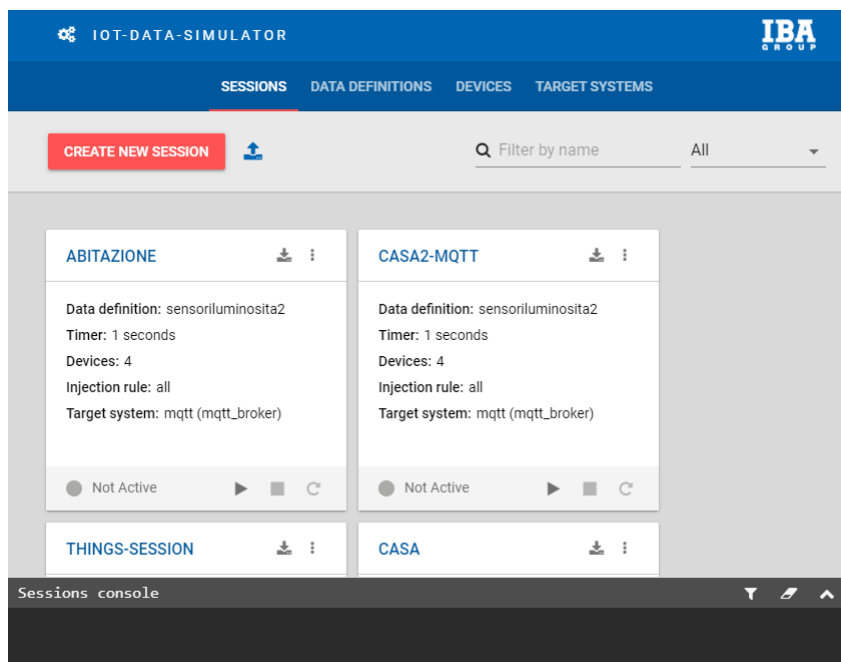


Figura 2.1: Interfaccia simulatore

Osservando l'interfaccia grafica ci si rende conto che è possibile accedere a 4 sezioni principali:

- Sessions
- Data Definitions
- Devices
- Target System

le quali corrispondono a i 4 concetti principali che abbiamo introdotto e spiegato nella sezione precedente.

Segue una descrizione dettagliata delle singole sezioni.

2.3.2 Session

Questa prima sezione corrisponde all'Homepage, se così possiamo chiamarla, a cui veniamo rediretti appena accediamo al simulatore.

Infatti, come mostrato in Figura 2.1, qui abbiamo la possibilità di creare una nuova sessione, cliccando sul tasto *Create New Session*, caricare una sessione esistente all'interno del File System ed inoltre ci vengono mostrare tutte le sessioni precedentemente create.

Ogni sessione presenta una sottosezione dedicata dove:

- vengono mostrate le informazioni riguardanti la sessione corrente e lo stato in cui si trova.
- vengono mostrati tre bottoni per, in ordine da destra a sinistra, avviare, fermare e fare il restart della sessione.
- viene data la possibilità di "scaricare la sessione", ovvero verrà scaricato un file *.json* con tutte le informazioni riguardanti la stessa.
- viene data la possibilità di modificare la sessione.

Inoltre in basso è presente una console dove verranno mostrati possibili *warning* o messaggi di errore, ma soprattutto dove verranno mostrati i dati generati dal simulatore in virtù delle regole che sono state definite.

2.3.3 Data Definitions

Accendo a questa sezione la prima pagina che viene presentata è la seguente:

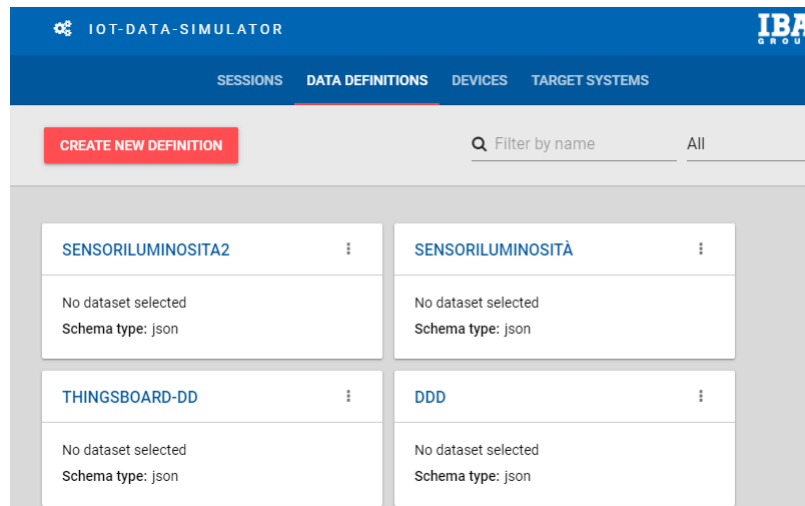


Figura 2.2: Data Definitions

Anche qui abbiamo la possibilità di creare un nuovo *Data Definitions* cliccando sul pulsante associato.

Inoltre anche qui abbiamo una visione generale di tutte le definition che abbiamo creato, avendo per ognuna la possibilità di modificarla.

Cliccando sul bottone *Create new definitions* accediamo ad un'altra pagina, in cui dobbiamo per prima cosa scegliere una tra queste opzioni:

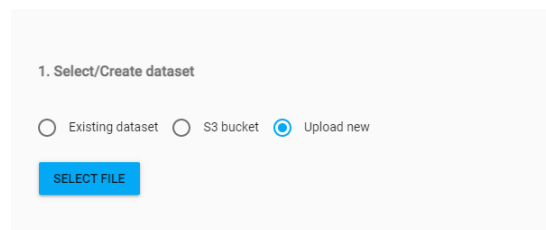


Figura 2.3:

In particolare:

- **Existing Dataset.** Permette di utilizzare un dataset caricato precedentemente tramite l'opzione *Upload new*. Tuttavia se si sceglie

quest'opzione, ma non si seleziona il dataset da utilizzare, allora questo verrà considerato come la volontà di non inserire un Dataset e in, accordo con quanto scritto nella sezione *2.3 Concetti fondamentali*, sarà necessario inserire uno Schema.

- **Upload new.** Permette di caricare un dataset da locale come file *.json* ad esempio.
- **S3 bucket.** Permette di caricare un file da cloud specificando l'url associato al file, in particolare utilizza il servizio Amazon S3.

A questo punto in base alle scelta riguardante l'utilizzo di un dataset ci troviamo davanti a due scenari:

- Se si decide di non inserire il Dataset allora verremo reindirizzati alla pagina seguente:

The screenshot shows a web interface for creating a schema. At the top, there's a tab labeled '2. Create schema or' and a button 'COPY FROM DEFINITION'. Below this, there's a form with two main sections: 'schema' and 'type'. The 'schema' section has a dropdown menu set to 'json'. The 'type' section has a dropdown menu set to 'object'. To the right of these dropdowns is a button labeled 'ADD PROPERTY'. Below these sections, there's a table with two rows, each representing a property. The first row has columns for 'type' (double), 'name' (brightnessDetected), 'property' (brightnessDetected), and 'description' (description). The second row has columns for 'type' (string), 'name' (sensorName), 'property' (sensorName), and 'description' (description). Each row has a red 'X' icon in the rightmost column, indicating an error or a required field.

type	name	property	description
double	brightnessDetected	brightnessDetected	description
string	sensorName	sensorName	description

Figura 2.4:

Dove sarà possibile creare lo Schema tramite il costruttore dell'interfaccia grafica.

- Se invece si decide di inserire un Dataset allora verremo comunque indirizzati alla pagina di cui sopra, ma ci ritroveremo con il Dataset già caricato. A questo punto sarà possibile definire o no uno Schema e salvare il Data Definition.

2.3.4 Device

La prima pagina a cui si viene reindirizzati appena si accede alla sezione risulta essere molto simile alla precedente in quanto fornisce le stesse funzionalità e la stessa struttura delle precedenti.

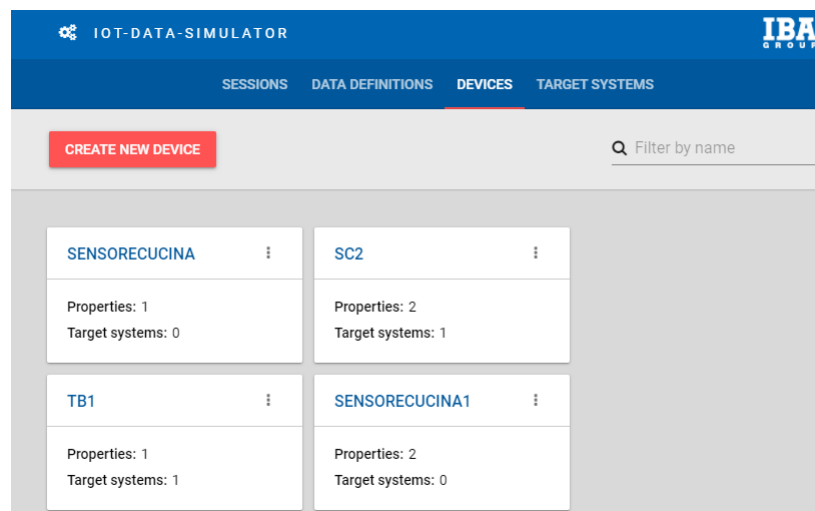


Figura 2.5: Devices

Cliccando sul bottone *Create new device* il simulatore dà la possibilità all'utente di creare un nuovo dispositivo specificando, oltre al nome del dispositivo, anche:

- i diversi attributi associati al dispositivo.
- i possibili sistemi di destinazione a cui i dati verranno inviati. Questa funzionalità risulta essere molto utile nel caso in cui diversi dispositivi debbano inviare le informazioni a sistemi diversi.

Device Injection

In relazione ai dispositivi, in fase di creazione di una sessione oltre alla semplice creazione, il simulatore fornisce un'altra funzionalità molto importante.

In particolare permette di configurare e definire la regola secondo cui debbano essere resi disponibili i dati dei diversi dispositivi.

La regola di iniezione presenta tre diverse possibilità:

- **Random.**

In uno scenario in cui abbiamo diversi dispositivi, ognuno con i propri dati, il simulatore sceglierà randomicamente un dispositivo tra quelli disponibili e i dati ad esso associati verranno forniti in output.

- **All.**

Questa opzione presenta due varianti:

- **Senza ritardo.**

Utilizzare l'opzione *All* lasciando il ritardo a 0 millisecondi porta il simulatore a rendere disponibili i dati di tutti i dispositivi ad ogni istante, in accordo con le regole di temporizzazione precedentemente definite. Ad esempio se abbiamo impostato che i dati devono essere disponibili ogni 5 secondi e abbiamo 4 dispositivi, allora ogni 5 secondi saranno disponibili tutti i dati di tutti i dispositivi.

- **Con ritardo.**

Affiancare un ritardo all'opzione *All* porta a simulare uno scenario in cui i dati dei singoli dispositivi non sono subito tutti disponibili ma ogni dispositivo inizia a fornire i propri dati dopo un certo tempo, pari al ritardo scelto, rispetto al dispositivo precedente. Per esempio se abbiamo tre dispositivi (*device1*, *device2* e *device3*) e abbiamo scelto un ritardo di 5 secondi. Allora *device1* inizierà a fornire i dati appena la sessione sarà messa in esecuzione, *device2* inizierà ad inviare dati solo dopo 5 secondi dall'inizio della sessione e *device3* inizierà ad inviare dati solo dopo 10 secondi dall'inizio della sessione.

- **Round Robin.**

Come suggerisce il nome questa opzione permette di iniettare i dispositivi secondo la schedulazione *Round Robin*.

2.3.5 Target System

Questa ultima sezione, che presenta un'interfaccia grafica molto simile alle precedenti:

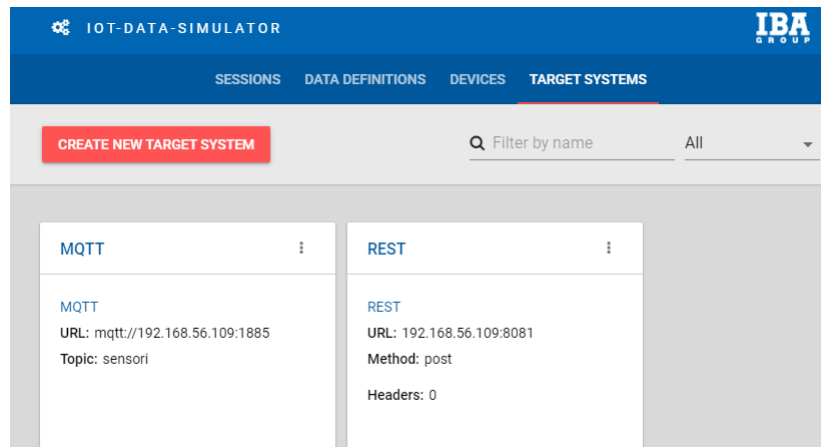


Figura 2.6: Target System

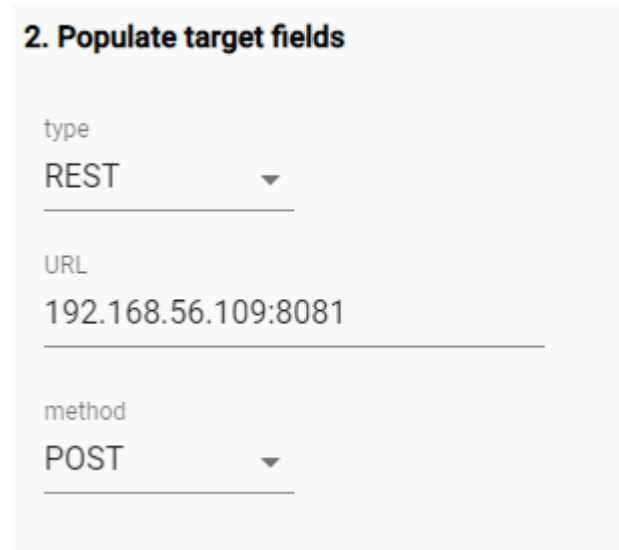
permette di definire uno o più sistemi di destinazione a cui inviare i dati.

La configurazione dei *Target System* varia in base al protocollo che si sceglie di utilizzare per la comunicazione, quindi ogni protocollo necessita di una serie di parametri diversi dagli altri.

Anche in questo caso il simulatore risulta essere molto flessibile e versatile in quanto supporta l'invio dei dati a numerosi sistemi di destinazione, tra cui:

- AMQP
- Kafka
- Local object storage (minio object storage)
- MQTT
- REST
- Websocket

Ad esempio un sistema di destinazione che utilizza il protocollo REST (HTTP) potrebbe essere configurato in questo modo:



The image shows a configuration window with the title "2. Populate target fields". It contains three input fields:

- A dropdown menu labeled "type" with the value "REST" selected.
- A text input field labeled "URL" containing the value "192.168.56.109:8081".
- A dropdown menu labeled "method" with the value "POST" selected.

Figura 2.7: Target System Setup

quindi specificando l'URL e la porta a cui eseguire una richiesta POST, considerando il caso descritto nell'immagine precedente.

2.4 Generazione dei dati

Dopo aver descritto dettagliatamente le diverse sezioni, l'ultimo aspetto di cui dobbiamo parlare è quello riguardante la generazione dei dati. Il simulatore fornisce diverse alternative riguardanti la generazione dei dati e le modalità di generazione cambiano da caso a caso, in virtù del sistema che si decide di simulare.

Infatti possiamo distinguere due casi principali:

- **Senza Dispositivi.**

Nel caso in cui si sia deciso di non simulare la presenza di dispositivi, in fase di definizione delle regole per la generazione dei dati, si hanno tre possibili opzioni:

- **Random.**

Disponibile solo nel caso di valori numerici, permette di specificare il range in cui i dati verranno randomicamente generati.

- **Literal.**

Permette di definire un valore statico per quel particolare attributo.

- **Custom Function.**

Permette di scrivere una funzione JavaScript che restituisca il valore di quel specifico attributo.

- **Con Dispositivi.**

Nel caso in cui si sia scelto di utilizzare dei dispositivi allora sarà possibile accedere agli attributi dei singoli dispositivi.

Capitolo 3

Il progetto

Questo terzo ed ultimo capitolo della trattazione sarà dedicato alla descrizione del progetto che sfrutta il simulatore descritto nel secondo capitolo.

Quindi verranno descritte tutte le fasi che hanno portato alla realizzazione del progetto in essere partendo dall'idea su cui quest'ultimo si basa fino ad arrivare alla descrizione dettagliata del codice che compone l'applicazione.

3.1 L'idea

In questa prima sezione verrà descritta rapidamente l'idea su cui si basa il progetto. Infatti si è cercato di sviluppare un progetto pensando ad uno scenario reale, introducendo tuttavia delle semplificazioni.

Lo scenario che è stato immaginato è quello di un'abitazione composta da 4 stanze come descritto nella figura seguente:

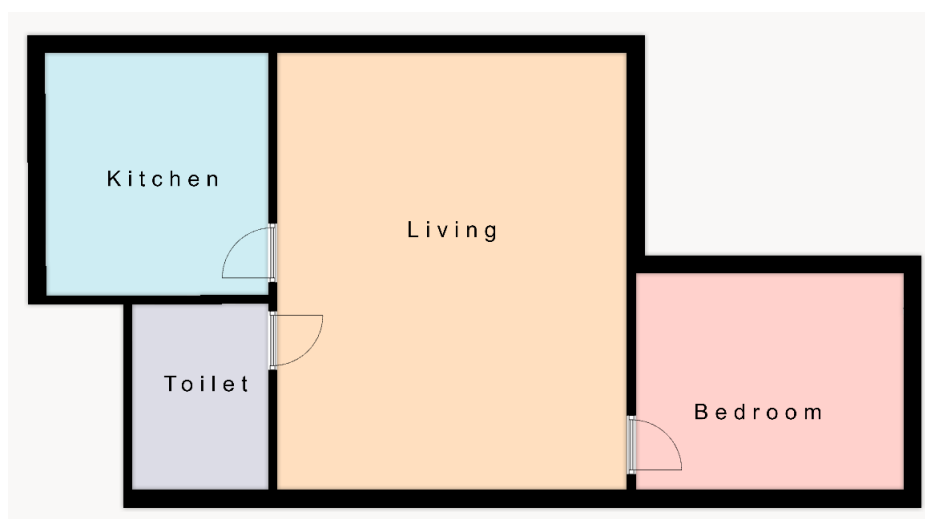


Figura 3.1: Schema Abitazione

In ogni stanza si è immaginato di installare un sensore che periodicamente rileva la luminosità all'interno della stanza e rende il dato disponibile.

Ogni volta che il dato è disponibile l'applicazione legge il dato, lo elabora e decide se cambiare o no lo stato delle luci della stanza associata, in modo tale che ogni stanza sia sempre ben illuminata.

3.2 Il funzionamento

Prima di andare a spiegare dettagliatamente i singoli componenti che compongono il progetto si è deciso di utilizzare questa sezione per spiegare il funzionamento dello stesso al fine di dare una panoramica generale. Per spiegare il funzionamento del progetto nel modo più facile e chiaro possibile facciamo riferimento alla figura sottostante:

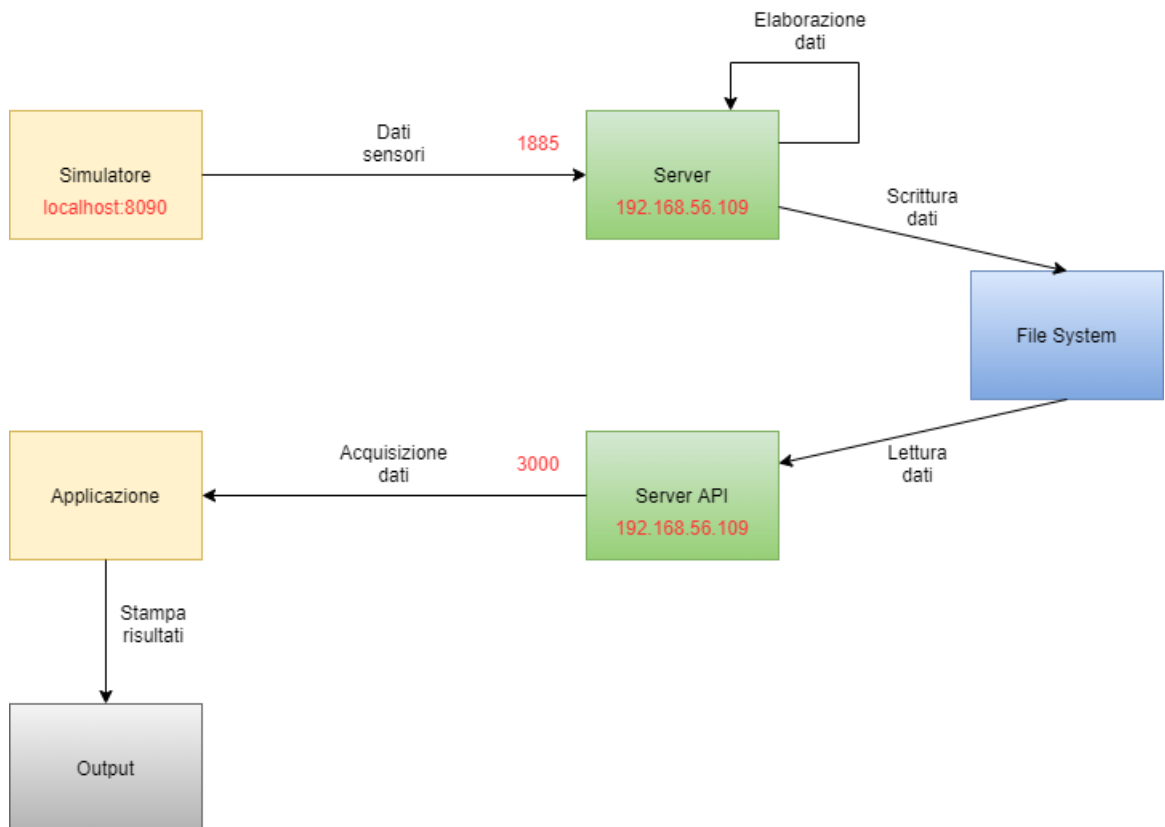


Figura 3.2: Schema Progetto

Osservando la figura possiamo distinguere 4 componenti principali:

- **Il Simulatore.**

Nel progetto il simulatore è stato utilizzato per simulare i sensori di luminosità. Infatti il simulatore ha il compito di generare i dati dei sensori ed inviarli al sistema esterno.

- **Il Server.**

I dati inviati dal simulatore vengono raccolti dal server in ascolto e salvati nel *File System*, in accordo con la logica interna del server stesso.

- **API.**

All'interno del progetto è presente anche un server che mette a disposizione delle REST API, con le quali è possibile accedere ai dati dei sensori salvati dal server principale.

- **L'Applicazione.**

L'ultimo componente dell'architettura è l'applicazione. Questa accede ai dati tramite le REST API e, in virtù dei dati ottenuti, decide se cambiare o no lo stato corrente delle luci.

Dopo questa piccola introduzione seguono 4 sezioni in cui saranno descritti dettagliatamente tutti i componenti che compongono l'architettura.

3.3 Il Simulatore

Questa prima sezione verrà dedicata alla descrizione di come è stato utilizzato e configurato il simulatore all'interno del progetto.

Come già anticipato nel paragrafo precedente il simulatore, all'interno del progetto, è stato utilizzato per simulare i sensori all'interno dell'abitazione. Per questo motivo è stata creata la sessione *Abitazione*, che si presenta in questo modo:

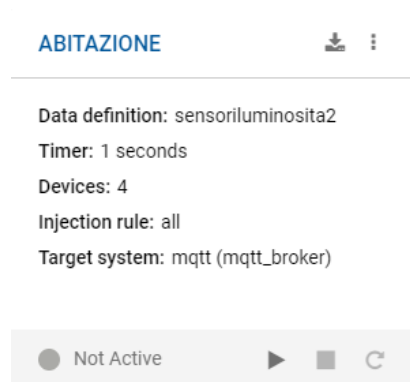


Figura 3.3: Sessione *Abitazione*

Per descrivere nel modo più dettagliato possibile le caratteristiche della sessione corrente si è deciso di riservare una sezione dedicata ad ogni componente della stessa.

3.3.1 Abitazione: Data Definition

Il primo componente di cui andiamo a parlare è quello riguardante la sezione del *Data Definition*.

In accordo con l'idea sulla quale si basa l'intero progetto, per questo componente è stato deciso di utilizzare un *Data Definition* che non presenta un Dataset, ma comunque presenta uno Schema Json. Infatti lo schema è stato creato con il costruttore messo a disposizione dall'interfaccia grafica e si presenta in questo modo:

The screenshot shows a web interface titled "2. Create schema". It features a form for defining a JSON schema. At the top, there are two dropdown menus: "schema" set to "json" and "type" set to "object". To the right of these is a button labeled "ADD PROPERTY". Below the dropdowns, there is a table with two rows, each representing a property. Each row has four columns: "type", "name", "property", and "description". The first row has "double" for type, "brightnessDetected" for name, "brightnessDetected" for property, and "description" for description. The second row has "string" for type, "sensorName" for name, "sensorName" for property, and "description" for description. To the right of each row is a red "X" icon, likely indicating a validation error or a required field.

type	name	property	description
double	brightnessDetected	brightnessDetected	description
string	sensorName	sensorName	description

Figura 3.4: Data Definition

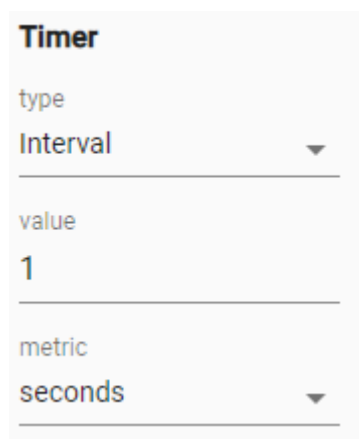
Quindi le *entry* generate dal simulatore si compongono di due parti:

- **brightnessDetected**, che rappresenta il livello di luminosità percentuale rilevato dal sensore associato alla stanza dell'abitazione.
- **sensorName**, che rappresenta il nome del sensore.

3.3.2 Abitazione: Regole temporali

Il secondo aspetto di cui andiamo a parlare riguarda le regole temporali adottate nella generazione dei dati.

La sezione dedicata alle regole temporali si presenta in questo modo:



The image shows a configuration window titled "Timer". It contains three fields, each with a label and a value, separated by horizontal lines. The first field is labeled "type" and has the value "Interval" with a downward arrow. The second field is labeled "value" and has the value "1". The third field is labeled "metric" and has the value "seconds" with a downward arrow.

Field	Value
type	Interval
value	1
metric	seconds

Figura 3.5: Regole temporali

Infatti, per il progetto, è stata scelta l'opzione *Interval* che, in questa configurazione, forza il simulatore a rendere disponibili i dati dei sensori ogni secondo. Ovviamente cambiando il campo *value* è possibile variare la frequenza di generazione.

3.3.3 Abitazione: Dispositivi

Sempre in accordo con l'idea su cui si basa il progetto si è deciso di creare 4 dispositivi, ognuno rappresentante un sensore all'interno dell'abitazione. Si è deciso di riportare, come esempio, il caso del sensore della cucina, che si presenta in questo modo:

The screenshot shows a two-step configuration interface for a device. Step 1, 'Populate name', has a text input field with the label 'name' and the value 'sensorecucina'. Step 2, 'Populate properties', features an 'ADD PROPERTY' button and a table with three columns: 'name', 'type', and 'value'. The table contains one row with the values 'sensorName', 'String', and 'sensoreCucina'. A trash icon is located to the right of the table.

name	type	value
sensorName	String	sensoreCucina

Figura 3.6: Esempio: Sensore Cucina

Infatti ogni sensore presenta un unico attributo, ovvero il suo nome, al quale sarà possibile nella sezione dedicata alle regole di generazione dei dati.

3.3.4 Abitazione: Regole di iniezione dei dispositivi

Dato che si è scelto di utilizzare dei dispositivi, è stato necessario configurare il simulatore anche per quanto riguarda le regole di iniezione degli stessi.

Riguardo a questo, come mostra la figura seguente:

The screenshot shows a configuration screen titled 'Step 4 of 7. Select device injection rule'. It contains two fields: a dropdown menu labeled 'type' with the value 'All' selected, and a text input field labeled 'delay (milliseconds)' with the value '0'.

Figura 3.7: Regole di iniezione dei dispositivi

si è scelto di utilizzare la funzione *All* con ritardo nullo, che ricordando simula il fatto che i sensori siano subito tutti accessi all'avvio della sessione, e quindi subito in grado di fornire dei dati.

3.3.5 Abitazione: Regole di generazione dei dati

Una volta definite le regole di iniezione dei dispositivi è ora di definire le regole riguardanti la generazione dei dati.

Come mostra la figura seguente:

Step 5 of 7. Data processing rules

schema	type	name	Rule type	Depends on												
json	object		Custom function													
<table border="1"> <thead> <tr> <th>type</th> <th>name</th> </tr> </thead> <tbody> <tr> <td>double</td> <td>brightnessDetected</td> </tr> <tr> <td colspan="2">property</td> </tr> <tr> <td>brightnessDetected</td> <td>description</td> </tr> </tbody> </table>			type	name	double	brightnessDetected	property		brightnessDetected	description	<div>OPEN EDITOR</div>					
type	name															
double	brightnessDetected															
property																
brightnessDetected	description															
<table border="1"> <thead> <tr> <th>type</th> <th>name</th> </tr> </thead> <tbody> <tr> <td>string</td> <td>sensorName</td> </tr> <tr> <td colspan="2">property</td> </tr> <tr> <td>sensorName</td> <td>description</td> </tr> </tbody> </table>			type	name	string	sensorName	property		sensorName	description	<table border="1"> <thead> <tr> <th>Rule type</th> <th>property name</th> </tr> </thead> <tbody> <tr> <td>Device property</td> <td>sensorName</td> </tr> </tbody> </table>		Rule type	property name	Device property	sensorName
type	name															
string	sensorName															
property																
sensorName	description															
Rule type	property name															
Device property	sensorName															

Figura 3.8: Regole di generazione dei dati

gli attributi da gestire, in accordo con il Dataset definito in precedenza sono due:

- **sensorName.**

Come anticipato, dato che si è scelto di simulare la presenza di dispositivi, in questa fase è possibile selezionare l'opzione *Device Property* e quindi accedere al nome dei singoli dispositivi per questo attributo.

- **brightnessDetected.**

Rispetto all'attributo precedente, in questo caso la regola per la

generazione dei dati risulta essere più complessa e quindi si è deciso di scrivere una sezione dedicata per descriverla al meglio.

Generazione dei valori associati al livello di luminosità

Dal Capitolo 2 emerge che per valori numerici il simulatore permette di generare numeri in modo randomico. Tuttavia, guardando il codice sorgente del simulatore disponibile su GitHub, ci si rende conto che è presente la classe *MathUtil.java* all'interno della quale sono presenti tutti i metodi utilizzati dal simulatore per la generazione randomica dei dati.

In particolare, osservando il metodo per la generazione dei Double, ci si rende conto che utilizza il metodo *.nextDouble()* della classe *Random* messa a disposizione da *java.util*.

In accordo con la documentazione Java riguardo il metodo in essere [7], tale metodo genera valori randomici seguendo una distribuzione uniforme.

Questo porta alla generazione di un flusso di valori che risultano essere anche molto distanti tra di loro. Per questo motivo una distribuzione di questo tipo non può essere utilizzata all'interno del progetto, in quanto è impossibile pensare che nel giro di pochi secondi la luminosità di una stanza passi ad esempio da un valore pari al 40% ad un valore del 90%. Quindi, per rendere la generazione dei dati il più verosimile possibile, si è scelto di utilizzare l'opzione *Custom Function* che permette di creare una funzione personalizzata per la generazione dei valori.

La funzione in essere si presenta in questo modo:

```
function custom(ruleState, sessionState, deviceName) {
    var hasSpare= false;
    var mean=50, stdDev=3;
    var spare;
    if (hasSpare) {
        hasSpare = false;
        return spare * stdDev + mean;
    } else {
        var u, v, s,temp;
        do {
            u = Math.random() * 2 - 1;
            v = Math.random() * 2 - 1;
            s = u * u + v * v;
        } while (s >= 1 || s === 0);
        temp = Math.sqrt(-2.0 * Math.log(s) / s);
        spare = v * temp; // z1=v*temp
        hasSpare = true;
        return mean + stdDev * u * temp; // z0= u*temp
    }
}
```

Infatti, al fine di generare dei dati verosimili, si è scelto di implementare una funzione che restituisce comunque valori casuali che però seguono una distribuzione normale. In questo modo specificando un valor medio ed una deviazione standard otterremo dei valori non troppo distanti tra di loro, risolvendo così il problema di generare sequenze di dati poco verosimili.

Al fine di generare dei valori che seguano una distribuzione normale è stata implementata una funzione che utilizza la forma polare della *Trasformata di Box-Muller*, chiamata anche *Metodo Polare Marsaglia*[11]. Il *Metodo Polare Marsaglia* è un metodo di campionamento di numeri pseudo-casuali per generare una coppia di variabili casuali normali stan-

dard indipendenti.

L'applicazione del metodo si divide in due fasi principali:

1. Si scelgono punti casuali

$$(u, v)$$

nel quadrato

$$-1 < u < 1, -1 < v < 1$$

finché

$$0 < s = u^2 + v^2 < 1$$

2. Trovata la coppia

$$(u, v)$$

che soddisfa la condizione di cui sopra, allora i due numeri casuali normalmente distribuiti sono del tipo:

$$Z_0 = u \sqrt{\frac{-2 \ln s}{s}}$$

,

$$Z_1 = v \sqrt{\frac{-2 \ln s}{s}}$$

Tuttavia questi numeri seguono la distribuzione normale standard, quindi con valor medio nullo e variazione standard pari a 1, per questo motivo l'ultima parte della funzione serve per ottenere dei valori che rispettino il valor medio e la variazione standard specificati.

3.3.6 Abitazione: Sistemi di destinazione

Per concludere il discorso riguardante l'utilizzo del simulatore all'interno del progetto l'ultima sezione che ci resta da descrivere è quella riguardante il sistema di destinazione.

La scelta del sistema di destinazione dipende da quale protocollo si vuole utilizzare per la trasmissione dei dati.

In questa fase si è scelto di utilizzare come protocollo di comunicazione il protocollo *MQTT*, in quanto risulta essere il protocollo maggiormente

utilizzato per le applicazioni Internet Of Things.

A tal proposito la sezione dedicata al protocollo si presenta in questo modo:

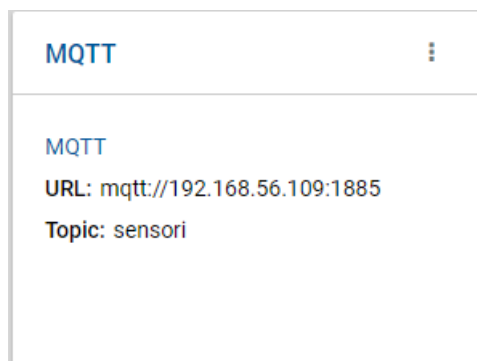


Figura 3.9: Sistema di destinazione MQTT

Sulla base della configurazione in essere il simulatore invierà i dati generati all'URL, alla porta specificata e renderà disponibili quest'ultimi sul topic *sensori*.

3.4 La raccolta dei dati

In accordo con lo schema presentato all’inizio di questo capitolo, questa sezione verrà dedicata alla descrizione dettagliata del componente utilizzato per la ricezione ed elaborazione dei dati.

3.4.1 Formato dei dati

In virtù delle scelte fatte durante la configurazione del simulatore il formato dei dati che vengono generati dal simulatore si presentano in questo modo:

```
2021-09-13 16:23:54 Abitazione Session has been started
2021-09-13 16:23:54 Abitazione {"brightnessDetected":27.28051928058298,"sensorName":"sensoreBagno"}
2021-09-13 16:23:54 Abitazione {"brightnessDetected":40.24832771353688,"sensorName":"sensoreSala"}
2021-09-13 16:23:54 Abitazione {"brightnessDetected":48.43714909737962,"sensorName":"sensoreCamera"}
2021-09-13 16:23:54 Abitazione {"brightnessDetected":43.35174629091131,"sensorName":"sensoreCucina"}
```

Figura 3.10: Formato dei dati

Quindi all’avvio della sessione *Abitazione* il simulatore, ogni secondo, genererà le quattro entry riportate in figura che stanno a rappresentare le quattro rilevazioni fatte dai sensori e si occuperà di inviare queste al sistema esterno.

3.4.2 Breve introduzione al protocollo MQTT

Al fine di comprendere al meglio il funzionamento del componente adibito alla ricezione dei dati inviati dal simulatore si è deciso di dedicare questa sezione per spiegare brevemente come funziona il protocollo MQTT.

MQTT (acronimo di *Message Queue Telemetry Transport*) è un protocollo applicativo che utilizza TCP/IP pensato per gestire le connessioni *Machine to Machine* (M2M).

A differenza del protocollo HTTP, il quale adotta un’architettura Client-Server basata sul paradigma *request-response*, il protocollo MQTT si basa

su un'architettura **Publisher and Subscriber**.

Osservando al figura seguente:

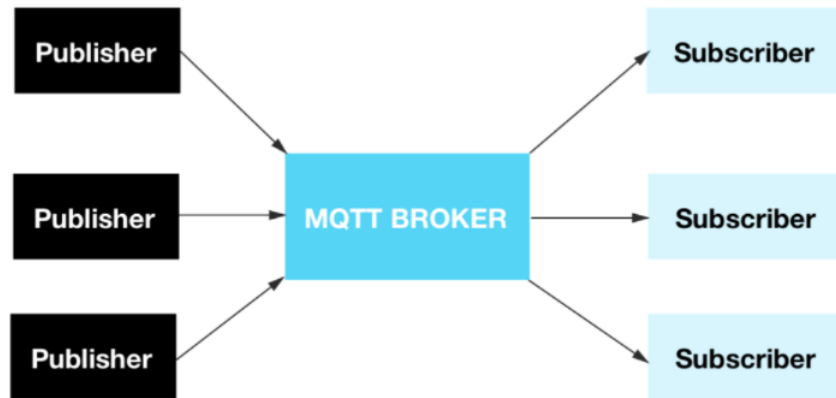


Figura 3.11: Schema MQTT

ci si rende conto che il protocollo MQTT è un protocollo:

- **Molti-a-Molti.** In quanto possono essere presenti numerosi *Publisher* e *Subscriber*.
- **Asincrono.** In quanto *Publisher* e *Subscriber* sono fortemente disaccoppiati tra di loro ed è compito del *Broker* gestire lo scambio di messaggi.

Sempre in accordo alla figura precedente, le entità in gioco sono:

- **Publisher.**

All'interno dell'architettura i *Publisher* hanno il compito di rendere disponibili ai *Subscriber* i messaggi che vogliono comunicare tramite il *Broker*. In particolare i *Publisher* pubblicano i messaggi su dei *Topics*, gestiti in modo gerarchico, a cui i *Subscriber* sono iscritti.

- **Subscriber.**

I *Subscriber*, invece, hanno il compito di iscriversi ai diversi *Topic*, tramite il *Broker*. In questo modo potranno accedere solo ai messaggi a cui sono interessati.

- **Broker.**

Ultimo componente dell'architettura, il *Broker* ha il compito di gestire lo scambio dei messaggi tra Publisher e Subscriber. In particolare, come già anticipato, il Broker gestisce la mole dei messaggi tramite l'utilizzo dei Topic. In questo modo quando e se un Publisher pubblica un messaggio su un particolare Topic, sarà compito del Broker gestire il messaggio e renderlo disponibile a tutti i Subscriber iscritti a quel topic.

3.5 Implementazione del protocollo MQTT

Una volta introdotto il protocollo MQTT siamo perfettamente in grado di comprendere come è stata gestita la raccolta e la gestione dei dati prodotti dal simulatore.

Al fine di descrivere al meglio come è stata implementata questa parte, facciamo riferimento allo schema corrente, che introduce maggiori dettagli rispetto allo schema generale all'inizio del capitolo:

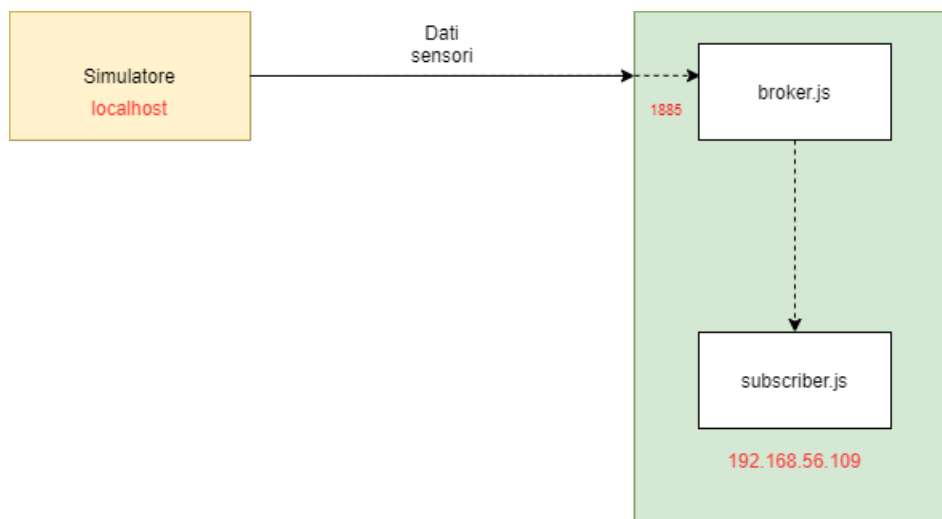


Figura 3.12: Schema Invio e Ricezione dei dati

Dalla figura emergono tre entità principali:

- **Il Simulatore**, il quale si comporta come un *Publisher* in quanto pubblica i messaggi contenenti i dati dei sensori sul topic *sensori*.

- **broker.js.**

Il Broker, come suggerisce la figura, è in esecuzione sull'indirizzo 192.168.56.109 ed è accessibile alla porta 1885.

Il codice sorgente del broker è molto semplice e si presenta in questo modo:

```
var mosca = require('mosca');
var settings = {
  port:1885
}

var server = new mosca.Server(settings);

server.on('ready', function(){
  console.log("ready");
});
```

Il codice del broker utilizza il modulo *Mosca*[12]. Questo è un modulo per *Node.js* che permette in poche righe di codice di configurare un broker MQTT con impostazioni personalizzate.

- **subscriber.js.**

Il codice sorgente del Subscriber si presenta in questo modo:

```
var mqtt = require('mqtt')
var client = mqtt.connect('mqtt://192.168.56.109:1885')
var fs = require("fs");
var fileName='/home/sec/Desktop/tesi/sensori_mqtt'
var message_counter=+0;
var current_detection='['
client.on('connect', function () {
  client.subscribe('sensori')
})

client.on('message', function (topic, message) {
  context = message.toString();
  console.log('message:'+context)
```

```
    if(message_counter <=18){
        current_detection+=context +','+'\n'
    }else if(message_counter==19){
        current_detection+=context +'\n'
    }
    message_counter++;

    if(message_counter===20){
        current_detection+=']'
        console.log('5 rilevazioni fatte')
        fs.writeFile(fileName, current_detection,
            function(err) {
                if(err) {
                    return console.log(err);
                }
                console.log("The file was saved!");
            });

        current_detection='['
        message_counter=0

    }

});
```

Il Subscriber funziona in questo modo:

1. Per prima cosa, tramite il modulo node *mqtt* inizializza un client MQTT per poter accedere ai dati dei sensori tramite il Broker. Infatti la riga `var client = mqtt.connect('mqtt://192.168.56.109:1885')` serve proprio per connettersi al Broker.
2. Dato che i dati sono pubblicati dal simulatore (il Publisher) sul topic *sensori*, il Subscriber, tramite il metodo `client.subscribe`, si iscrive al topic specificato.

3. A questo punto inizia la vera ricezione dei dati. Infatti ogni volta che viene pubblicato un messaggio, ovvero ogni volta che il simulatore invia una rilevazione da parte dei sensori, il Subscriber legge il messaggio e concatena il messaggio alla stringa *current_detection*, formattando la stringa in formato *.json*.
4. Raggiunte le 5 rilevazioni, quindi 20 messaggi ottenuti (questo deriva dal fatto che abbiamo 4 sensori), scrive il contenuto della stringa *current_detection* sul file *sensori_mqtt*, e si rimette in ascolto di successivi messaggi.

In questo modo, ogni circa 5 secondi, abbiamo un file *.json* che contiene 5 quartine del tipo:

```
{"brightnessDetected":66.29037765528692,"sensorName":"sensoreBagno"},  
{"brightnessDetected":41.5296968536518,"sensorName":"sensoreSala"},  
{"brightnessDetected":42.945445028858344,"sensorName":"sensoreCamera"},  
{"brightnessDetected":45.91858901781233,"sensorName":"sensoreCucina"},
```

in cui ogni entry rappresenta la rilevazione di luminosità corrente fatta da un sensore.

3.6 API

Questa sezione sarà dedicata alla descrizione del componente che mette a disposizione le API per poter accedere ai dati acquisiti dal simulatore. L'implementazione delle API è stata fatta tramite l'utilizzo di *Next.js*[13], un framework di React che mette a disposizione tutta una serie di funzionalità, incluse quelle riguardanti la creazione delle API.

3.6.1 Creazione del progetto

Al fine di utilizzare le funzionalità mette a disposizione da *Next.js*, la prima cosa da fare è l'inizializzazione del progetto tramite il comando `npx create-next-app`. Infatti lanciando il comando da terminale vengono caricati tutti i moduli necessari al funzionamento e la cartella del progetto si presenta in questo modo:

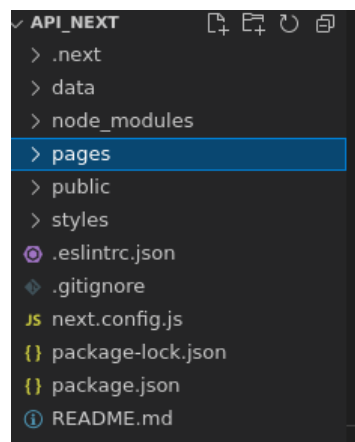


Figura 3.13: Progetto API

In questo modo sarà possibile mettere in esecuzione il componente e sarà disponibile all'URL `192.168.56.109:3000`.

3.6.2 Implementazione delle API

La figura precedente evidenzia la cartella *pages*. Infatti, al fine di creare delle API personalizzate, dobbiamo andare a lavorare proprio su

quella cartella.

In particolare all'interno di quest'ultima è stata creata una sotto-cartella *api*, la quale contiene a sua volta la cartella *sensori* che contiene gli script *.js* che implementano le funzionalità delle API.

All'interno della cartella *pages/api/sensori* sono presenti due script:

- **index.js.**

Il codice sorgente di questo script si presenta in questo modo:

```
const fs = require('fs')
export default function(req,res){
  const sensori =
    JSON.parse(fs.readFileSync('/home/sec/Desktop/tesi/sensori_mqtt'));
  res.status(200).json(sensori)
}
```

Questa API viene invocata facendo una richiesta all'URL *192.168.56.109:3000/api/sensori* e restituisce l'intero contenuto del file *sensori_mqtt*.
facendo una richiesta di prova tale indirizzo otteniamo:

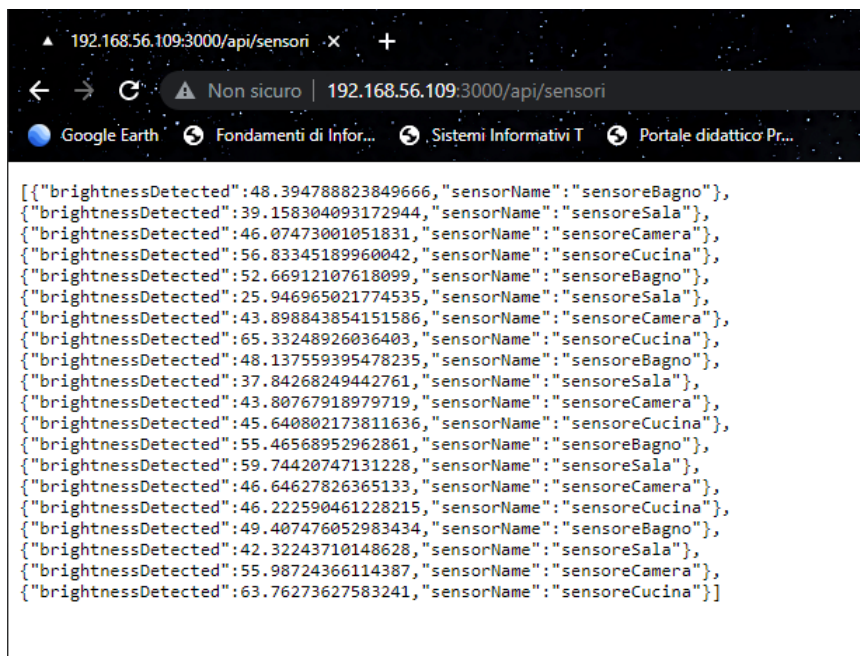


Figura 3.14: index.js output

- [sensorName].js.

Il codice sorgente di questo script si presenta in questo modo:

```
const fs = require('fs')

export default function handler(req,res){
  const { sensorName } = req.query
  const sensori =
    JSON.parse(fs.readFileSync('/home/sec/Desktop/tesi/sensori_mqtt'));
  const sensore = sensori.filter((item) =>
    item.sensorName === sensorName)
  res.status(200).json(sensore)
}
```

Questa API viene invocata facendo una richiesta all'URL *192.168.56.109:3000/api/sensori/<nome_sensore>* e restituisce solo le rilevazioni inerenti al sensore specificato.

Facendo una richiesta di prova a tale, specificando "sensoreBagno", l'output che otteniamo è il seguente:

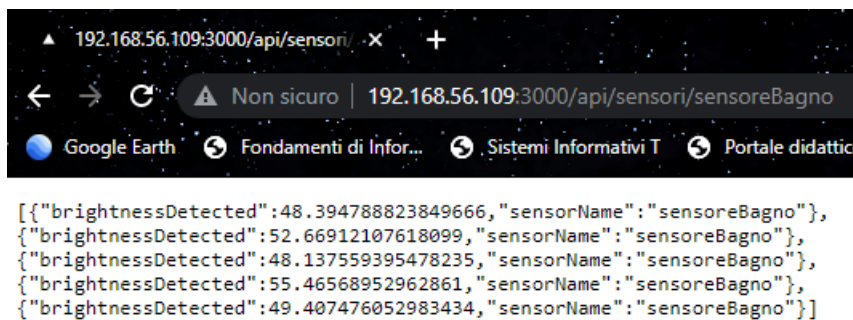


Figura 3.15: [sensorName].js output

3.7 L'applicazione

Questa ultima sezione della trattazione sarà dedicata alla descrizione del funzionamento dell'applicazione *SmartLight*.

In accordo con lo schema ad inizio capitolo, il compito dell'applicazione è quello di acquisire i dati dei sensori, elaborare tali dati, decidere se cambiare o no lo stato delle luci e restituire lo stato corrente delle luci in output.

Al fine di descrivere al meglio l'intera applicazione si è scelto di seguire il flusso delle chiamate delle funzioni all'interno del *main*.

3.7.1 Fase 1: Caricamento della configurazione

Il *main* dell'applicazione si trova all'interno della classe *LightController.java*, la quale presenta questi attributi:

```
public String address;  
public String port;  
public String base_route;  
public ObjectMapper mapper = new ObjectMapper();
```

Fatta eccezione per l'attributo *mapper* che ci servirà più avanti per spiegare come è avvenuto la scrittura e la lettura dei file *.json*, gli altri attributi servono all'applicazione per poter utilizzare le API descritte nella sezione precedente. Infatti, il primo metodo che viene invocato all'interno del *main* è il metodo *LightController.init()*, che si presenta in questo modo:

```
public void init(String configFilePath) throws  
    FileNotFoundException, IOException  
{  
    Properties appProps = new Properties();  
    appProps.load(new FileInputStream(configFilePath));  
    address = appProps.get("address").toString();  
    port = appProps.get("port").toString();  
    base_route = appProps.get("base.route").toString();
```

```
}
```

Tale metodo, tramite l'utilizzo della classe *Properties*, inizializza gli attributi della classe con i parametri specificati all'interno del file di configurazione che viene passato come parametro alla funzione e come argomento all'applicazione. Il file *.properties* si presenta in questo modo:

```
address=192.168.56.109
port=:3000
base.route=/api/sensori
```

Quest'ultimo contiene tutti i parametri necessari al fine di poter utilizzare le API implementate. Ovviamente sarebbe stato possibile inserire tali dati direttamente all'interno del codice, ma si è deciso di non farlo per prevedere possibili modifiche future.

3.7.2 Fase 2: Acquisizione delle rilevazioni dei sensori

Una volta caricata la configurazione ed acquisiti i parametri necessari, l'applicazione è perfettamente in grado di chiamare le API e ottenere i dati dei sensori.

Per questo motivo, all'interno della classe *LightController* è stato implementato il metodo *callAPI()*. Questo si presenta in questo modo:

```
public Sensore[] callAPI() throws IOException {

    String line=null;
    String url_api="http://" + this.address + this.port +
        this.base_route ;

    //API call
    URL url = new URL(url_api);
    HttpURLConnection conn = (HttpURLConnection)
        url.openConnection();
```

```
conn.setRequestMethod("GET");
conn.setRequestProperty("Accept", "application/json");

// obtain API response
InputStream responseStream = conn.getInputStream();
StringBuilder builder= new StringBuilder();
BufferedReader reader = new BufferedReader(new
    InputStreamReader(responseStream));
while((line=reader.readLine()) != null) {
    builder.append(line);
}

//parse API response
String textJson= builder.toString();
JsonNode actualObj=mapper.readTree(textJson);
String currentJson=
    mapper.writerWithDefaultPrettyPrinter().writeValueAsString(actualObj);
//System.out.println(currentJson);
Sensore[] sensori = mapper.readValue(currentJson,
    Sensore[].class);

return sensori;
}
```

Il metodo funziona in questo modo:

1. Per prima cosa prepara l'URL associato alle API.
2. Tramite la classe *URLConnection* esegue una richiesta GET all'URL specificato.
3. Tramite la classe *BufferedReader* legge, riga per riga, la risposta delle API.
4. Tramite la libreria *Jackson*, mappa le informazioni contenute nella risposta (ovvero le rilevazioni dei sensori) in un array di *Sensore* e

restituisce tale array.

Infatti la classe *Sensore* che viene utilizzata presenta solo 2 attributi: **sensorName** e **brightnessDetected**.

3.7.3 Fase 3: Caricamento dello stato delle luci

In virtù del fatto che l'applicazione ha il compito di cambiare lo stato delle luci in relazione ai dati attenuti dai sensori, sarà necessario per prima cosa caricare lo stato corrente delle luci.

Per questo motivo è stata implementata la classe *Lampade.java* che presenta il metodo *loadLightStatus()*. Tramite l'invocazione di questo metodo viene letto il file *luci.json*, che si presenta in questo modo:

```
[{"name": "luceSala", "status": "spenta"},  
{"name": "luceBagno", "status": "accesa"},  
{"name": "luceCucina", "status": "accesa"},  
{"name": "luceCamera", "status": "spenta"}]
```

Tale file contiene 4 entry ognuna rappresentante lo stato di una luce, quindi se sarà necessario cambiare lo stato di una luce verrà modificato questo file. La necessità di utilizzare un file per mantenere lo stato delle luci risiede nel fatto che le luci, all'avvio dell'applicazione, sono in uno stato non noto a priori.

3.7.4 Fase 4: Elaborazione dei dati e stampa dei risultati

Quest'ultima sezione è quella più complessa e lunga in quanto verrà descritta la logica su cui si basa l'applicazione.

Seguendo il flusso delle operazioni invocate all'interno del *main*, finalmente, arriviamo all'invocazione del metodo *updateLight()*. Il metodo si presenta in questo modo:

```
public void updateLight(String lightFile, Sensore[]  
    sensori, Lampade lampade ) throws JsonProcessingException,  
    IOException {
```

```
String[]
    correctStatus=this.checkCorrectStatus(sensori,lampade);

//leggo il file json delle lampade e modifico lo stato
//delle stesse in virtu dei valori contenuti dentro
//correctstatus rispettando la sequenza
//Sala-Bagno-Cucina-Camera
List<Lampada> temp= new ArrayList<>();

temp =
    Arrays.asList(mapper.readValue(Paths.get(lightFile).toFile(),
        Lampada[].class));

//setto i nuovi stati
temp.set(0, new Lampada("luceSala",correctStatus[0])); //
//sala
temp.set(1, new Lampada("luceBagno",correctStatus[1]));
//bagno
temp.set(2, new Lampada("luceCucina",correctStatus[2]));
//Cucina
temp.set(3, new Lampada("luceCamera",correctStatus[3]));
//Camera

//modifico il file luci.json introducendo i nuovi stati
//delle luci,quindi scrivo sul file luci.json
mapper.writeValue(new File(lightFile), temp);

}
```

Il metodo presenta tre parametri in ingresso:

- **String lightFile**, contiene il percorso del file *luci.json*.
- **Sensore[] sensori**, contiene le rilevazioni fatte dai sensori ottenute precedentemente con la *callAPI()*.
- **Lampade lampade**, che contiene lo stato attuale delle luci caricato con la *loadLightStatus()*

All'interno del metodo sono presenti numerose chiamate a metodi di supporto quindi si è deciso di dedicare una sezione ad ogni operazione effettuata.

Fase 4.1: Controllo dello stato delle luci

Al fine di capire se cambiare o non lo stato delle luci, il primo metodo che viene invocato è *checkCorrectStatus()*. Tale metodo esegue queste operazioni:

1. Per prima cosa, controlla se all'interno delle stanze c'è un livello di luminosità consono che per semplicità è stato impostato al 50% per ogni stanza.

Al fine di portare avanti questo compito invoca, per ogni stanza, il metodo *isDark<nome_stanza>()*. Ad esempio nel caso del bagno il metodo si presenta in questo modo:

```
public boolean isDarkBathroom(Sensore[] sensori)
    throws JsonProcessingException, IOException {

    boolean isDark=false;
    double average=0;

    List<Sensore>
        sensoriBagno=this.filterSensor("sensoreBagno",
            sensori);

    average= this.getMedia(sensoriBagno);
    System.out.println("Valore Minimo Bagno: 50%
        ---->"+ " Valore Corrente Bagno: "+ average +"%");
    if(average >=50)
        isDark=false;
    else
        isDark=true;

    return isDark;
```

```
}
```

Al metodo viene passato come parametro l'array contenente le rilevazioni dei sensori, ed effettua in ordine queste operazioni:

- (a) Invoca il metodo *filterSensor()*, il quale seleziona, dall'array contenente tutte le rilevazioni dei sensori, e restituisce una lista contenente solo le rilevazioni associate al nome del sensore passato come parametro.
 - (b) Esegue la media, tramite il metodo *getMedia()*, dei valori assunti dal parametro *brightnessDetected*.
 - (c) Infine, in virtù del valore della media calcolato, restituisce un boolean che se *true* indicata che nella stanza è buio e quindi sarà opportuno accendere la luce, mentre se *false* indica che il livello di luminosità della stanza è sopra la soglia e quindi la luce dovrà essere spenta.
2. A questo punto, per ogni singola stanza, viene eseguita questa porzione di codice che, per il caso della Sala, si presenta in questo modo:

```
String
    statoCalcolato=this.switchStatus(isDarkLivingRoom,
    lampade.getLuci().get(0).getStatus());

String
    statoPrecedente=lampade.getLuci().get(0).getStatus();

    correctStatus[0]=statoCalcolato.equalsIgnoreCase("NO_SWITCH"?
    statoPrecedente : statoCalcolato;
```

Questa porzione di codice esegue queste operazioni:

- (a) Per prima cosa invoca il metodo *switchStatus()*. A questo vengono passati come parametri il boolean risultante dalla chiamata del metodo *isDark<nome_stanza>* e una stringa

rappresentante lo stato corrente della luce all'interno della stanza specificata. In virtù dei parametri ottenuti il metodo *switchStatus()* restituisce una stringa che può assumere 3 valori:

- **accesa**, che indica che la luce associata alla stanza corrente deve essere accesa.
- **spenta**, che indica che la luce associata alla stanza corrente deve essere spenta.
- **NO_SWITCH**, che indica che la luce associata alla stanza corrente si trova nello stato corretto, quindi se è spenta deve rimanere spenta e lo stesso vale per il caso in cui sia accesa.

- (b) Ottenuto lo stato corretto delle luci questo viene inserito all'interno di un array che contiene gli stati corretti di tutte le luci presenti all'interno dell'abitazione.

Fase 4.2: Modifica dello stato delle luci

La conclusione dell'esecuzione della Fase 4.1 ci porta a conoscere lo stato corretto in cui le luci devono trovarsi al fine di rispettare i livelli di luminosità richiesti (50% per ogni stanza). Quindi a questo punto quello che ci resta da fare è modificare lo stato delle luci, ovvero modificare il file *luci.json*.

Infatti, continuando a descrivere il codice di *updateLight()*, vengono eseguite queste operazioni:

1. Viene letto il file *luci.json* e salvato lo stato corrente delle luci all'interno di una lista temporanea.
2. Viene modificata la lista con gli stati corretti.
3. Viene modificato il file *luci.json* con gli stati aggiornati.

Fase 4.3: Stampa dei risultati

Conclusa anche la fase di elaborazione dei dati, tramite il metodo *printLightstatus()* l'output che abbiamo in uscita, ad ogni esecuzione, è quello mostrato nella figura seguente:

```
Valore Minimo Bagno: 50% ----> Valore Corrente Bagno: 49.68837626301368%
Valore Minimo Cucina: 50% ----> Valore Corrente Cucina: 51.60701183715982%
Valore Minimo Sala: 50% ----> Valore Corrente Sala: 48.48548982587691%
Valore Minimo Camera: 50% ----> Valore Corrente Camera: 50.46987216642764%
Stato delle luci prima dell'analisi dei sensori:
Lampada(name=luceSala, status=spenta)
Lampada(name=luceBagno, status=spenta)
Lampada(name=luceCucina, status=accesa)
Lampada(name=luceCamera, status=spenta)
Stato delle Luci dopo l'analisi dei sensori:
Lampada(name=luceSala, status=accesa)
Lampada(name=luceBagno, status=accesa)
Lampada(name=luceCucina, status=spenta)
Lampada(name=luceCamera, status=spenta)
```

Figura 3.16: Output Applicazione

Quindi, come si vede in figura, per ogni stanza viene mostrato lo stato delle luci prima e dopo l'esecuzione dell'applicazione. Infatti dall'immagine si vede come, in virtù del valore di luminosità rilevato, l'applicazione decide o no se cambiare lo stato delle luci.

3.8 Test del Progetto

In questa ultima sezione della trattazione si è scelto di inserire i risultati dei test effettuati sui diversi componenti che compongono il progetto.

3.8.1 Test subscriber.js

Il primo test che si è scelto di effettuare è quello riguardante il componente per l'acquisizione dei dati (*subscriber.js*).

In particolare dopo 20 interazioni i risultati ottenuti sono questi:

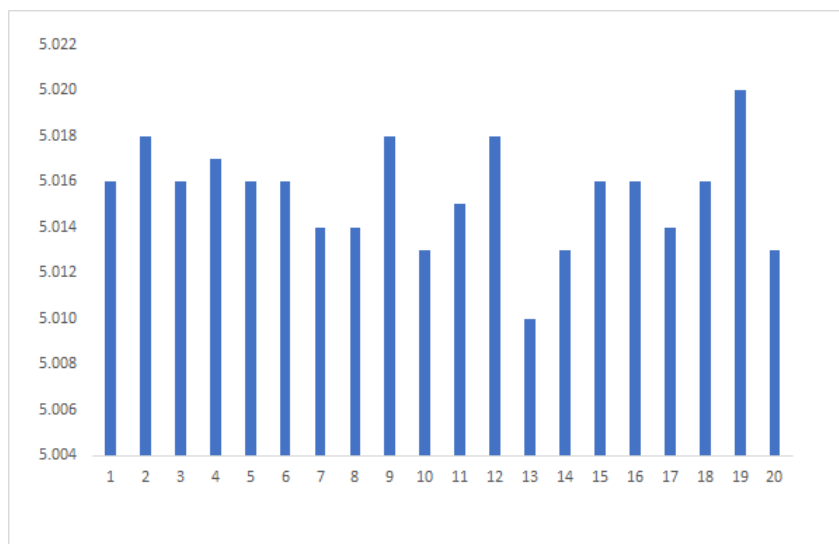
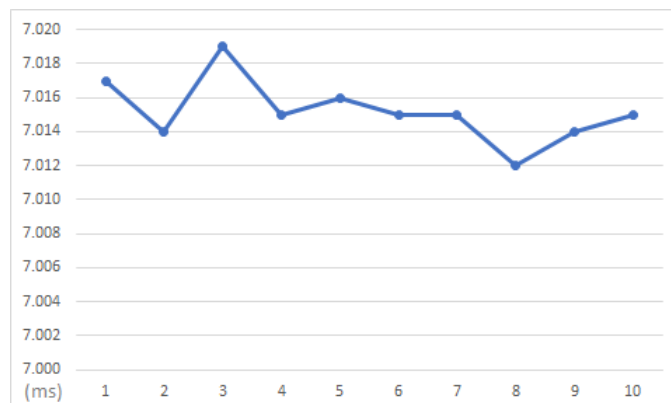


Figura 3.17: Tempo di esecuzione *subscriber.js*

Infatti dal grafico è possibile vedere che il *subscriber.js* rende disponibili i dati dei sensori, cioè scrive il file *sensori_mqtt.json*, ogni 5 secondi circa.

3.8.2 Test SmartLight

Il secondo componente che è stato testato è l'applicazione *SmartLight*. In questo caso si è scelto di eseguire l'applicazione 10 volte e i risultati sono i seguenti:

Figura 3.18: Tempo di esecuzione *SmartLight*

Osservando il grafico si capisce che *SmartLight* impiega circa 7 secondi per ogni esecuzione. In realtà bisogna tenere in considerazione che l'applicazione è stata forzata a rimanere in attesa per 7 secondi, dopo ogni iterazione, in modo tale da lasciare il tempo al *subscriber.js* e al server delle API di rendere disponibili i dati dei sensori.

Conclusioni

L'*Internet Of Things*, al giorno d'oggi, è forse uno degli argomenti su cui si discute maggiormente. Infatti con questo elaborato si è cercato di spiegare, nel modo migliore possibile, tutto quello che c'è dietro a questo vasto argomento cercando di far emergere tutte le sfaccettature del paradigma *Internet Of Things*.

Come abbiamo già anticipato all'interno della trattazione, forse il vantaggio più grande che emerge dallo studio dell'argomento è la sua forte versatilità, in quanto oggi giorno tutti i dispositivi hanno una connessione internet e quindi potenzialmente possono far parte di un ecosistema in cui interagiscono con altri dispositivi.

Inoltre durante lo sviluppo dell'applicazione SmartLight ho potuto utilizzare le funzionalità messe a disposizione dal simulatore *IoT-Data-Simulator* conoscendone anche i limiti. Infatti parlando dei vantaggi e degli svantaggi del simulatore possiamo dire che:

1. Come già anticipato, essendo un simulatore generico e quindi non legato a particolari tecnologie lascia molto spazio alla fantasia. Infatti risulta molto versatile ed adattabile a diversi utilizzi, in quanto permette di simulare sistemi complessi contenenti numerosi dispositivi e lascia all'utente la possibilità di personalizzare completamente le regole per la generazione dei dati.
2. Altro vantaggio, sicuramente non scontato, è la facilità di installazione. Infatti per mettere in esecuzione in simulatore basta scaricarlo dalla documentazione fornita e metterlo in esecuzione tramite *Docker*.

3. Infine un altro vantaggio è la facilità di utilizzo, in quanto è possibile utilizzarlo a scatola chiusa scegliendo la configurazione semplicemente tramite l'interfaccia grafica.

Tuttavia nell'utilizzo sono emerse anche delle limitazioni. Infatti forse la più grande limitazione che ho riscontrato durante la mia esperienza di utilizzo, almeno da quello che ho potuto osservare, è l'impossibilità del simulatore di accettare dati in ingresso. Infatti, il simulatore permette di inviare dati a sistemi esterni ma non di ricevere dati. Ad esempio, a causa di questa limitazione, all'interno del progetto ho dovuto gestire le luci con un file esterno.

Concludendo posso dire comunque che il simulatore IoT-Data-simulator si è comunque dimostrato uno strumento utile e valido al fine di costruire piccole applicazioni IoT.

Bibliografia e Sitografia

- [1] In: (). URL: <https://www.mckinsey.com/featured-insights/internet-of-things/our-insights>.
- [2] «ANASYS». In: (). URL: www.ansys.com/it.
- [3] Colliers. «An interview with Nikola Tesla by John B. Kennedy». In: (gen. 1926).
- [4] «IBA-Group-IT/IoT-data-simulator». In: (apr. 2018). URL: <https://github.com/IBA-Group-IT/IoT-data-simulator>.
- [5] «IDC FutureScape: Worldwide Digital Transformation 2021 Predictions». In: (ott. 2020). URL: <https://www.idc.com/getdoc.jsp?containerId=prUS46967420>.
- [6] internet4things.it. In: (). URL: <https://www.internet4things.it/iot-library/internet-of-things-gli-ambiti-applicativi-in-italia/>.
- [7] «Javadoc». In: (). URL: <https://docs.oracle.com/javase/8/docs/api/java/util/Random.html>.
- [8] R. Khan, S. Khan, R. Zaheer e S. Khan. «Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges». In: dic. 2012, pp. 257–260. ISBN: 978-1-4673-4946-8. DOI: 10.1109/FIT.2012.53.
- [9] «La storia dell'IoT: una cronologia completa dei principali eventi». In: (lug. 2018). URL: <https://hqsoftwarelab.com/>.
- [10] K. L. Lueth. «Perché l'Internet of Things si chiama Internet of Things: definizione, storia, disambiguazione». In: (dic. 2014).

-
- [11] «Metodo polare Marsaglia». In: (). URL: https://en.wikipedia.org/wiki/Marsaglia_polar_method.
 - [12] «Mosca Documentation». In: (lug. 2018). URL: <https://github.com/moscajs/mosca#readme>.
 - [13] «Next.js». In: (). URL: <https://nextjs.org/docs/api-routes/introduction>.
 - [14] smartmeway.com. In: (). URL: <https://www.smartmeway.com/press/all-press/48-internet-of-things.html>.
 - [15] trends.google.com. *Google Trends*. 2021. URL: <http://trends.google.com/trends>.

Ringraziamenti

Vorrei ringraziare, per prima cosa, il professore Bellavista Paolo per aver seguito pazientemente tutto lo sviluppo di questo elaborato e avermi dato preziosi consigli.

Ringrazio mio padre Stefano e mia madre Simonetta perché mi sono sempre stati accanto e non mi hanno mai fatto mancare il loro sostegno e il loro aiuto durante questi anni.

Ringrazio tutti gli amici e le amiche di Narni con cui sono rimasto in contatto anche essendo lontano da casa e che mi hanno accolto ad ogni mio rientro a casa.

Ringrazio Solfrizzi, GBully, luzzi99 e L40, conosciuti in Università e con i quali ho seguito su Discord tutte le lezioni durante l'emergenza Covid-19. Infine un ringraziamento speciale va a Elena con cui ho iniziato questa avventura a Bologna. Lei è la persona che meglio conosce tutto quello che è successo in questi anni e per questo la voglio ringraziare di tutto il sostegno e l'affetto che mi ha dato in questi 4 anni.