Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Interoperabilità dell'IoT con Web of Things e Generative AI

Relatore:
Dott.
Luca Sciullo

Presentata da:
Simone Tassi

Sessione III
Anno Accademico 2022-2023

Dedicato ai miei nonni

Abstract

Negli ultimi anni tecnologie come l'*Internet of Things* e l'*Intelligenza Artificiale* hanno assunto un ruolo sempre più centrale nella vita dell'uomo. Gli standard del W3C *Web of Things* proposti di recente, hanno permesso di attenuare i problemi di interoperabilità, che da sempre caratterizzano il mondo IoT.

I nuovi modelli di *IA Generativa* hanno trovato recentemente applicazione nei processi di sviluppo software, grazie alla loro capacità di generare codice, automatizzando compiti ripetitivi e accelerando la creazione di prototipi.

In questa tesi viene illustrato il processo implementativo di un'applicazione Android che, grazie all'unione di IoT, WoT e Intelligenza Artificiale Generativa, crea delle automazioni eseguibili sia su dispositivo mobile che su Edge, a partire da una descrizione data dall'utente.

La possibilità di eseguire automazioni su due piattaforme diverse risulta essere vantaggioso poiché riduce le limitazioni che caratterizzano ognuna di esse e permette all'utente di avere un maggior controllo sulle proprie risorse.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

L'obiettivo principale dell'*Internet of Things* (IoT) è quello di consentire la comunicazione tra diversi dispositivi, sensori e attuatori (*Things*) attraverso la rete Internet. Dalla sua nascita, quando i dispositivi connessi erano pochi e molto semplici, ad oggi, l'IoT ha visto un grosso aumento dei campi a cui può essere applicato, come quello domotico, industriale e sanitario.
L'elevato numero di devices IoT si traduce in una frammentazione nelle tecnologie e nei protocolli utilizzati per la loro progettazione. Questo porta a un problema di interoperabilità che limita le potenzialità dell'Internet of Things. Diverse sono le soluzioni proposte dalla ricerca al problema dell'interoperabilità, una tra tutte è quella del *Web of Things*. Questo paradigma prevede l'utilizzo di tecnologie Web già conosciute e diffuse per costruire un layer superiore a quello dell'IoT, dove tutte le Things possono dialogare utilizzando un unico sistema di comunicazione basato su architetture RESTful. Il *World Wide Web Consortium* (W3C) nel 2014 ha espresso interesse verso questa tecnologia e da allora si impegna ad elaborare una serie di standard, rendendo il processo di sviluppo del WoT ancora più veloce.
Un altro campo informatico che, come quello dell'Internet of Things, è in forte crescita è quello dell'Intelligenza Artificiale. Sono sempre più numerosi gli strumenti che basano il loro funzionamento sull'AI Generativa, come chatbots e generatori di immagini, video e testo.

Nonostante i progressi fatti negli ultimi anni nel campo dell'*Internet of Things* e del *Web of Things*, la creazione di *mashup applications* (applicazioni in grado di ottenere dati da diverse Things ed elaborarli) risulta tuttora un'operazione complessa e da eseguire manualmente. Richiede quindi una buona padronanza degli strumenti IoT e WoT [23][34].

Lo scopo di questa tesi è quello di creare, tramite l'utilizzo del *Web of Things* e dell'*Intelligenza Artificiale*, un'applicazione Android in grado di generare automaticamente, a partire da una serie di istruzioni fornite dall'utente (non esperto), delle automazioni più o meno complesse che coinvolgano diverse Things.

L'elaborato è strutturato come segue:

- nel capitolo 2 viene fatta un'analisi dello stato dell'arte delle tecnologie utilizzate nello sviluppo del progetto, ovvero *Internet of Things, Web of Things* e *Intelligenza Artificiale Generativa*, soffermandosi anche sul concetto di *Interoperabilità*;
- il capitolo 3 presenta le caratteristiche architetturali, i requisiti e gli obiettivi dell'applicazione sviluppata;
- nel capitolo 4 sono analizzate le scelte implementative e sono illustrate le tecnologie utilizzate;
- nel capitolo 5 si misura l'efficienza e la precisione dell'applicazione attraverso dei casi d'uso.

Capitolo 2

Stato dell'arte

2.1 Internet of Things

L'Internet of Things (IoT) è una rete di oggetti e dispositivi fisici (things) in grado di connettersi e scambiare dati grazie a sensori, software e altre tecnologie che incorporano. Nasce a cavallo tra la fine degli anni '90 e l'inizio degli anni 2000. Da allora il numero di dispositivi connessi ha avuto una crescita esponenziale: ad oggi si contano più di 15 miliardi di things (quasi il doppio della popolazione mondiale) e si stima che entro il 2030 saranno circa 30 miliardi [39].
Il paese con il più alto numero di dispositivi IoT è la Cina, con 8 miliardi di things. Anche in Italia, dove si contano circa 124 milioni di dispositivi connessi [33], il mondo dell'IoT è in forte crescita, particolarmente nei settori Smart Car, Smart Metering e Smart Asset Management [32].

La prima definizione di IoT vedeva le things come oggetti identificabili in maniera univoca interconnessi tramite RFID (radio-frequency identification). Oggi, grazie al progresso tecnologico (disponibilità di microprocessori sempre più piccoli e potenti) e alla facilità di connessione ad Internet i dispositivi IoT assumono i ruoli più disparati, dalle lampadine di una Smart Home a sensori per il monitoraggio di pazienti affetti da Parkinson [14].

Indipendentemente dalle tecnologie utilizzate, con l'Internet of Things

viene introdotto un nuovo paradigma: la comunicazione *machine-to-machine*, che permette ai calcolatori di "*vedere, ascoltare, odorare il mondo autonomamente, [...] senza le limitazioni dell'inserimento manuale dei dati*" [5].

2.1.1 Architettura

Data la moltitudine di dispositivi differenti che si devono interfacciare tramite l'Internet of Things, il design della sua architettura deve essere scalabile ed estensibile. Non esiste un'architettura standard, ma la più utilizzata é quella a quattro strati [3][20].

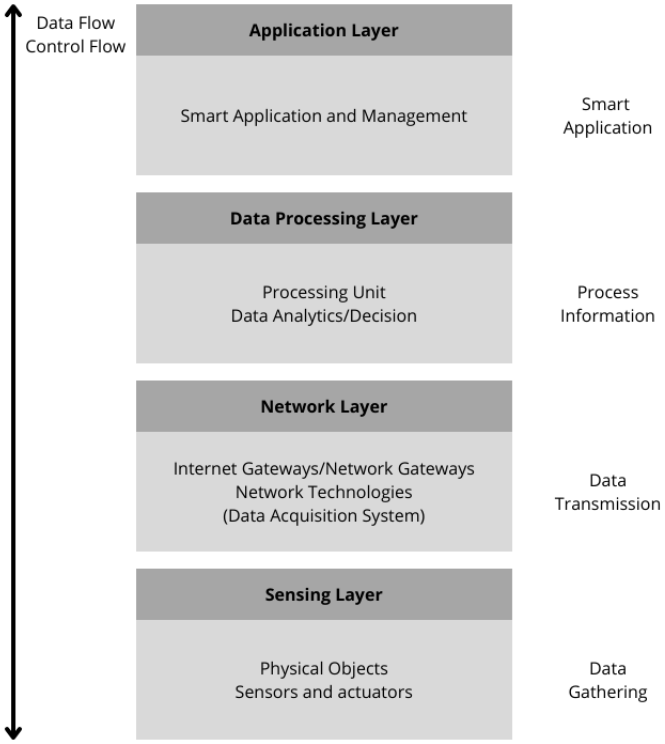


Figura 2.1: Architettura IoT a quattro strati

Sensing Layer

È lo strato più basso dell'architettura, costituito da sensori e attuatori che interagiscono con l'ambiente circostante. Questi dispositivi sono univocamente identificabili e sono in grado di fornire informazioni e ricevere comandi dal livello superiore, al quale sono connessi.

Network Layer

Gestisce la trasmissione attraverso la rete dei dati raccolti dal Sensing Layer. Include protocolli e tecnologie che abilitano la comunicazione tra dispositivi, come WiFi, Bluetooth, ZigBee, 4G e 5G. Può anche includere meccanismi crittografici e di autenticazione.

Data Processing Layer

Crea, cerca e gestisce i servizi che vengono richiesti da utenti e applicazioni, che vengono poi eseguiti sulla rete. Si occupa del *data storage* e permette l'interazione tra diverse things e servizi. All'interno di questo layer i dati sono analizzati e semanticamente riconosciuti, in modo da capire quale uso ne debba essere fatto.

Application Layer

Rappresenta lo strato superiore, con il quale l'utente interagisce. È responsabile dell'interfaccia e delle funzionalità di alto livello dell'ecosistema IoT. Include applicazioni mobile, web-app e altri metodi tramite i quali l'utente si può interfacciare all'infastruttura sottostante. In questo livello i dati sono ulteriormente processati e presentati al consumatore finale.

2.1.2 Dispositivi e protocolli di comunicazione

I dispositivi IoT possono essere di svariate tipologie, ma hanno tutti delle caratteristiche comuni. Sono oggetti fisici, solitamente sensori o attuatori,

che dispongono di un processore e di un firmware, spesso con capacità limitate alle azioni che essi devono deve eseguire. Sono dotati di una scheda di rete, elemento chiave che permette la comunicazione con gli altri dispositivi connessi all'Internet of Things, e anche di una memoria che permette di immagazziare ed elaborare informazioni [30].

La comunicazione tra dispositivi IoT, date le risorse limitate a cui essi hanno accesso, deve essere necessariamente a basso consumo e sono diversi i protocolli e le tecnologie utilizzabili. Si dividono in tecnologie di comunicazione di corto raggio e tecnologie di comunicazione di lungo raggio [36][37]:

- Tecnologie di comunicazione di corto raggio: la maggior parte delle applicazioni IoT prevedono la comunicazione tra things localizzate nello stesso ambiente o in ambienti vicini, sono infatti numerose le tecnologie che possono essere utilizzate in questo senso. Una tra tutte é la Radio Frequency Identification (RFID), tecnologia "madre" del mondo IoT. Oltre a quest'ultima anche altri metodi di identificazione come Barcode, QR Code e NFC (Near Field Communication) possono essere considerate tecnologie a corto raggio. Tra le più usate vi sono inoltre le tecnologie Bluetooth, ZigBee, Z-Wave e IPv6 Over Low Power WPAN (6LoWPAN).

- Tecnologie di comunicazione di lungo raggio: per l'interconnessione di diversi ambienti vengono utilizzate anche tecnologie a lungo raggio, quali: LoRa, SigFox, Ingenu, Symphony, LTE.

2.1.3 Casi d'uso

I casi d'uso dell'Internet of Things ad oggi sono numerosissimi, dall'automotive all'assistenza sanitaria, dalla logistica all'agricoltura.

Agricoltura

L'introduzione di sensori e dispositivi IoT permette agli agricoltori di mappare le proprie aree di coltivazione e ai ricercatori di condurre studi più approfonditi su temi come la fenotipizzazione o la qualità del suolo. Oltre a ciò l'impiego di smart things consente un maggiore controllo delle condizioni atmosferiche e quindi una migliore gestione delle risorse idriche. Anche nelle coltivazioni in serra l'uso di sensori è molto diffuso per la misurazione dei livelli di umidità, della temperatura e per la raccolta di informazioni sul suolo [2].

Assistenza sanitaria

In un campo vasto come quello della sanità gli esempi di applicazione di tecnologie IoT sono molti e diversificati. Molto diffuso é il monitoraggio del paziente attraverso diverse tipologie di sensori che alla minima anomalia sono in grado di avvertire il personale sanitario [2]. Dispositivi IoT sono anche utilizzati per il controllo della strumentazione medica e del suo funzionamento, oltre che per l'analisi della reazione del paziente a una determinata cura [6][13]. Sono diversi gli studi che negli ultimi anni si stanno occupando dell'utilizzo di tecnologie IoT per semplificare la vita quotidiana di persone con disabilità: dall'uso di un campanello smart che permette a una persona non udente di essere notificata una volta che il campanello viene suonato a applicazioni che permettono a persone non vedenti, tramite l'uso di tecnologie NFC e RFID, di fare la spesa autonomamente [4].

Industria e logistica

Sempre di più sono le macchine automatiche dotate di sistemi IoT che ne diagnosticano problemi e ne permettono una veloce riparazione, consentendo un'ottimizzazione delle risorse e uno snellimento delle pratiche all'interno della filiera produttiva [22]. Sensori IoT sono utilizzati anche per il controllo qualità dei prodotti prima, di metterli in commercio e per il monitoraggio

della parte logistica, una volta che il prodotto è pronto per raggiungere il consumatore [29][12].

Smart cities

L'inserimento in contesti urbani di dispositivi e sensori IoT é sempre più diffuso e permette una migliore gestione delle risorse energetiche, per esempio attraverso l'implementazione di sistemi di *smart lighting* che contribuiscono anche a ridurre l'inquinamento luminoso [9]. Sensori IoT sono utilizzati anche per il monitoraggio dell'integrità strutturale di edifici storici, oltre che per il controllo dei livelli di inquinamento atmosferico e acustico consentendo poi la stesura di un piano mirato per mitigarne gli effetti [31].

Smart homes

Nel contesto domestico le smart things sono ormai ampiamente diffuse e, oltre ad automatizzare lavori ripetitivi, permettono all'utente una gestione ottimizzata delle risorse e dei consumi [25]. Ad esempio un calorifero smart e una serie di termometri connessi alla rete, in autonomia possono regolare la temperatura della casa evitando sprechi.

2.2 Interoperabilità

Il termine *interoperabilità* è definito come "*il grado in cui due prodotti, programmi, ecc. possono essere utilizzati insieme, o la qualità di poter essere utilizzati insieme*" [24].
L'elevato numero di dispositivi IoT corrisponde a una considerevole frammentazione nelle tecnologie utilizzate e porta l'utente a doversi confrontare con problemi di interoperabilità che si traducono in maggiori costi di sviluppo e mantenimento delle infrastrutture. Sebbene dal punto di vista dei produttori di dispositivi IoT il problema dell'interoperabilità possa essere visto come un vantaggio, in quanto il consumatore spesso è costretto a scegliere una linea di prodotti offerti da un singolo fornitore e quindi a legarsi ad esso anche per

il futuro, la ricerca, sia in campo accademico che industriale si sta muovendo per elaborare nuove soluzioni.

I problemi di interoperabilità nel mondo IoT possono essere visti da diverse prospettive, per questo è necessario analizzarle una ad una [28][35]:

- **Interoperabilità tra dispositivi**: le differenze che ci sono tra tutti i dispositivi IoT (ad esempio tra un RaspberryPi e un tag RFID) e i diversi protocolli di comunicazione usati rendono difficile e non immediata la comunicazione tra essi.

- **Interoperabilità a livello rete**: si occupa di meccanismi per consentire lo scambio di messaggi senza soluzione di continuità tra sistemi attraverso reti diverse (reti di reti) per la comunicazione end-to-end. Affinché i sistemi siano interoperabili, ciascun sistema dovrebbe essere in grado di scambiare messaggi con altri sistemi attraverso vari tipi di reti. A causa dell'ambiente di rete dinamico ed eterogeneo nell'IoT, il livello di interoperabilità di rete dovrebbe gestire problemi come l'indirizzamento, il routing, l'ottimizzazione delle risorse, la sicurezza, la qualità del servizio e il supporto alla mobilità.

- **Interoperabilità sintattica**: si riferisce all'interoperabilità del formato e della struttura dati utilizzati nello scambio di informazioni o servizi tra entità di sistema IoT eterogenee. Il contenuto dei messaggi deve essere serializzato per essere inviato sul canale: il mittente codifica i dati in un messaggio utilizzando regole sintattiche specificate in una grammatica. Il destinatario decodifica il messaggio ricevuto utilizzando regole sintattiche definite nella stessa o in un'altra grammatica. I problemi di interoperabilità sintattica sorgono quando le regole di codifica del mittente sono incompatibili con le regole di decodifica del destinatario, causando un mismatch tra gli alberi di analisi del messaggio.

- **Interoperabilità semantica**: riguarda la capacità di agenti, servizi e applicazioni di scambiare informazioni, dati e conoscenze in mo-

do significativo, sia su Web che al di fuori di esso. Tuttavia, attualmente si ha una frammentazione, con dispositivi che spesso non parlano lo stesso linguaggio e non possono scambiarsi informazioni. Questa frammentazione è dovuta alla diversità nei modelli e nei data-schema utilizzati dai dispositivi IoT. Ciò porta a una incompatibilità semantica che impedisce ai sistemi IoT di interoperare dinamicamente e automaticamente.

- **Interoperabilità tra piattaforme**: la varietà di sistemi operativi, linguaggi di programmazione, strutture dati, architetture e meccanismi di accesso utilizzati nel mondo IoT ostacola lo sviluppo di applicazioni trasversali e cross-domain, poiché gli sviluppatori devono acquisire una conoscenza approfondita delle API specifiche della piattaforma e dei modelli informativi per adattare le proprie applicazioni. L'interoperabilità cross-platform consente alle applicazioni IoT di accedere e integrare dati da diverse piattaforme, facilitando la costruzione di applicazioni orizzontali IoT attraverso l'integrazione di piattaforme eterogenee. Questa interoperabilità è fondamentale per sviluppare soluzioni innovative che coinvolgono diversi domini IoT (smart home, assistenza sanitaria, trasporti, ...).

2.3 Web of Things

Il Web of Things (WoT) è un paradigma che nasce alla fine degli anni 2000 e si pone come obiettivo quello di eliminare i problemi di interoperabilità che hanno sempre caratterizzato il mondo IoT, mediante l'utilizzo delle tecnologie web [35].
La crescita esponenziale dei dispositivi IoT in tutto il mondo negli ultimi anni ha evidenziato come sia necessario intervenire sulla problematica della frammentazione. La ricerca sul Web of Thing è infatti in continuo sviluppo. Il WoT si configura come un paradigma complementare all'IoT, che permette di istanziare, a partire da una "thing", una "Web Thing", quindi accessibile

sul Web e oggetto di interazioni *web-based*. Il vantaggio principale che offre il Web of Things è la possibilità di interagire con le Web Things servendosi di standard già largamente utilizzati e conosciuti, (come JSON, HTTP, XML, ecc ...). Una Web Thing è quindi una vera e propria risorsa web alla quale è assegnato un URI e le cui funzionalità sono accessibili tramite una API RESTful.

Pur essendo sempre stato un tema molto seguito dalla ricerca sia in ambito industriale che accademico il Web of Things, solo negli ultimi anni ha subito un processo di standardizzazione, grazie al W3C (World Wide Web Consortium). Prima dell'intervento del W3C altre intepretazioni del WoT erano state fornite, tra le più significanti il Social Web of Things e il Semantic Web of Things [35].

Social Web of Things

Rappresenta la convergenza tra Web Sociale e Web of Things. Consente agli utenti di gestire, accedere, condividere e integrare smart things tramite piattaforme online (Social Network Sites) dove le persone pubblicano, collaborano e condividono informazioni con altri individui o gruppi, costruendo relazioni sociali.

Semantic Web of Things

Le tecnologie per il Web Semantico hanno giocato un ruolo fondamentale nel raggiungimento di una completa interoperabilità nel Web of Things, introducendo un modo per rendere i dati semanticamente ricchi e *machine-understandable*. L'obiettivo principale del Semantic WoT era quello di arrivare a definire una descrizione semantica universale per ogni Web Thing, semplice o più elaborata. Esso è stato raggiunto più recentemente grazie al lavoro di standardizzazione iniziato dal W3C.

2.3.1 Architettura

L'architettura del Web of Things, segue una struttura a strati, come l'architettura IoT. Ogni strato è indipendente e può essere implementato all'interno di un singolo dispositivo, all'interno di un gateway, o su più nodi di rete. L'architettura a layers più utilizzata è quella proposta da Guinard e Trifa [18] e può essere vista come un'estensione di uno stack di rete a cui si aggiungono layer applicazione WoT-related [27].

- **Access Layer**: permette a qualsiasi thing di diventare una Web Thing, fornendo quindi una API REST per poter interagire con il mondo esterno;
- **Find Layer**: consente alla Web Thing di essere trovata sulla rete da altri dispositivi usando le rappresentazioni standard del web;
- **Share Layer**: definisce come i dati prodotti dalla Web Thing debbano essere condivisi sulla rete, garantendo sicurezza ed efficienza. Le piattaforme Social WoT possono massimizzare la diffusione dei dati.
- **Compose Layer**: si occupa delle interazioni tra la Web Thing in questione e tutte le altre Web Things e servizi disponibili sulla rete.

Un altro approccio, differente dall'architettura a strati è quello dell'architettura *WT-centric*, nucleo del W3C Web of Things [35].

2.3.2 W3C Web of Things

Nel 2014 il World Wide Web Consortium si interessa al Web of Things e nel 2020, dopo diversi meetings e workshops vengono pubblicate le prime due W3C WoT Recommendations: WoT Thing Description e WoT Architecture [41].
Elemento centrale della concezione di Web of Things del W3C è la Thing: astrazione di un'entità fisica o virtuale descritta mediante una *Thing Description* (TD) in formato JSON, che fornisce le informazioni necessarie all'inte-

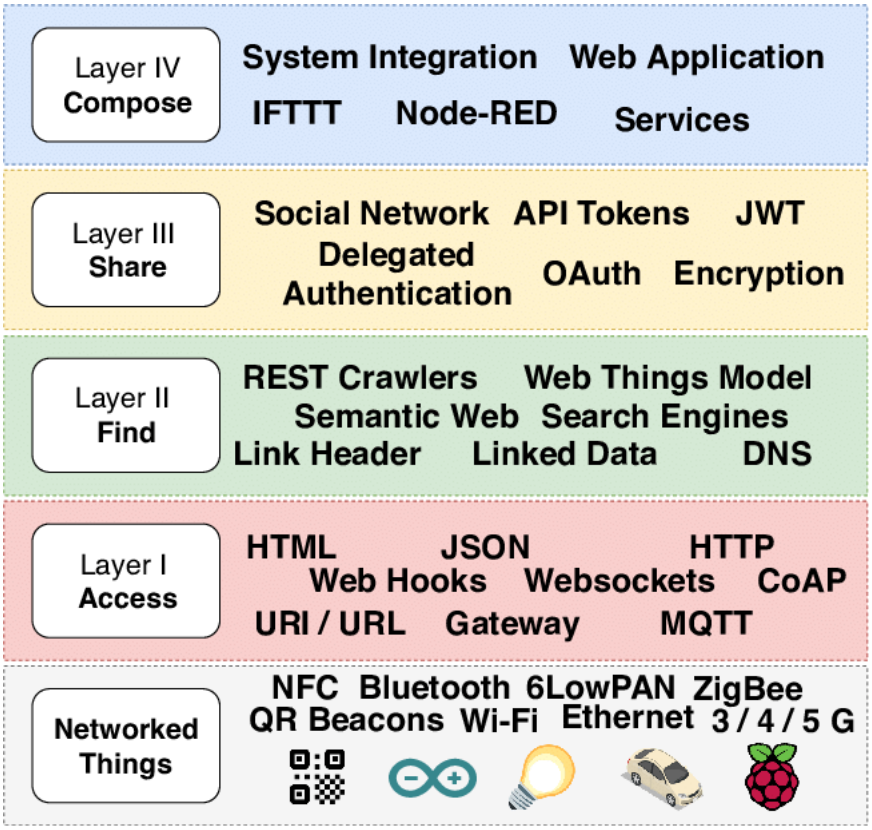


Figura 2.2: Architettura WoT a strati [7]

razione con la Thing. Essa è caratterizzata da cinque punti fondamentali [35] [43]:

- **Behavior**: rappresenta la logica di business della Thing e contiene le implementazioni sia dei suoi comportamenti automatici, sia delle interazioni che possono essere sviluppate su di essa;
- **Interaction Affordances**: definiscono quali sono le interazioni che l'utente può avere con la Thing, senza fare riferimento però a specifici modelli implementativi. Le Interaction Affordances possono essere distinte in *proprietà, azioni* ed *eventi*.
  - Proprietà: una *property* è uno stato della Thing che viene esposto e può essere letto, modificato e osservato (il suo valore viene ri-

tornato una volta subita una modifica). Una proprietà *read-only* può essere modificata attraverso un'azione apposita.

- Azioni: una *action* permette di invocare una funzione della Thing. Può manipolare uno stato della Thing che non è direttamente esposto o una o più proprietà. L'invocazione di una azione può anche avviare un processo di manipolazione dello stato, anche fisico (tramite attuatori), della Thing.
- Eventi: tramite gli *events* vengono comunicati i cambiamenti di stato della Thing. Gli stati non sono necessariamente esposti (vedi properties).

- **Data Schemas**: descrivono il modello delle informazioni e la struttura del payload correlata, nonché i dati corrispondenti che vengono scambiati tra le Things e i "Consumatori" durante le interazioni.

- **Security Configuration**: contiene i meccanismi di sicurezza per l'accesso alle *Interaction Affordances*. Include metadati di *public security* che descrivono i meccanismi di accesso alla Thing, ma non contengono dati sensibili, e *private security* che contengono invece dati sensibili.

- **Protocol Bindings**: corrispondono alla traduzione di una Interaction Affordance in un messaggio di uno specifico protocollo (e.g. HTTP, CoAP, ...). Servono al "Consumatore" per capire come interagire con la Thing tramite un'interfaccia di rete. Per supportare l'interoperabilità seguono la REST Uniform Interface e sono serializzati come *hypermedia controls*. Non tutti i protocolli di comunicazione sono adatti per l'implementazione di WoT Protocol Bindings.

Un altro elemento fondamentale del W3C WoT è quello dei *Binding Templates* [42]. Data la varietà di protocolli e tecnologie utilizzate nell'IoT è necessario creare templates che descrivano tutti gli elementi essenziali all'implementazione di un sistema di comunicazione. Una volta creato il *binding*

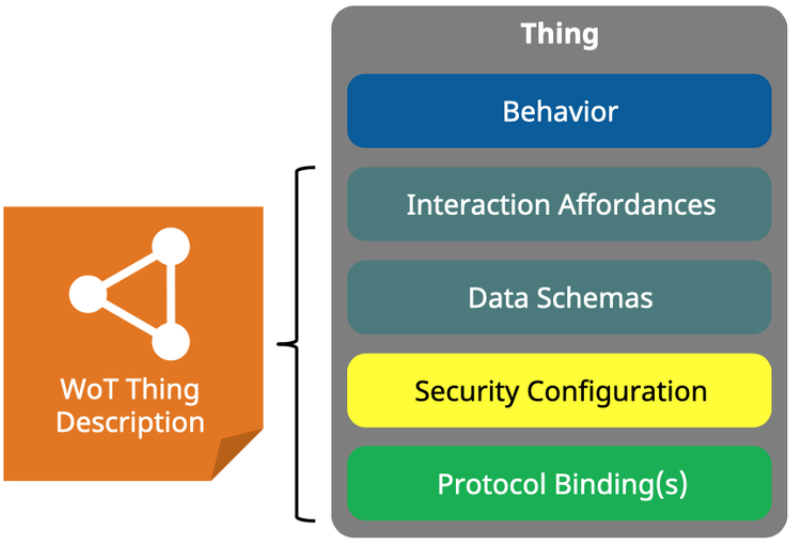


Figura 2.3: Thing Description [43]

template per una determinata tecnologia IoT, viene incluso all'interno della Thing Description ed è compito del "Consumatore" implementare i *protocol bindings* corrispondenti. Un Binding Template descrive bindings per *protocolli, payloads* e *piattaforme*:

- **Protocolli**: protocolli a livello applicazione come HTTP, COaP e MQTT.
- **Payloads**: diversi formati di payloads e media types che possono essere rappresentati all'interno della TD tramite Data Schemas e forms.
- **Piattaforme**: piattaforme e frameworks che combinano l'uso di determinati protocolli e payloads, descritti tramite TDs.

Un ulteriore elemento costitutivo della struttura del W3C Web of Things è la *Scripting API* [44] che fornisce una API ECMAScript-based simile alle API dei web-browsers. Integrando un sistema di scripting runtime nel WoT Runtime, la WoT Scripting API consente di utilizzare application-scripts che definiscono il comportamento di Things, Consumatori e Intermediari.

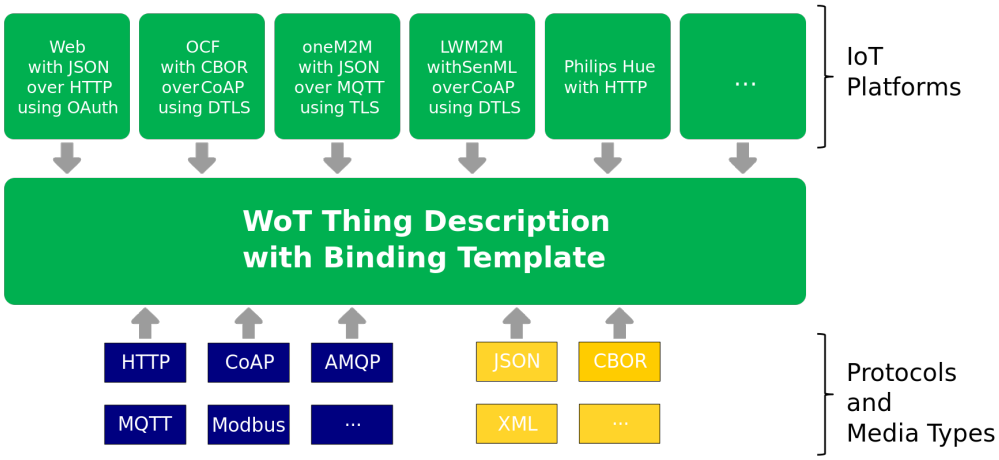


Figura 2.4: Da Binding Templates a Protocol Bindings [42]

La comunicazione tra componenti del WoT può essere *diretta* o *indiretta*. In ogni caso per parlare di comunicazione è necessario introdurre il concetto di *WoT Servient*, stack software che implementa tutte le componenti fin'ora descritte [43].

Un Servient può ospitare ed esporre una Thing oppure la può consumare, può quindi agire sia da server che da client, nel caso la comunicazione sia diretta, oppure da intermediario, in caso di comunicazione indiretta. Spetta al *WoT Runtime*, integrato nel Servient, istanziare la rappresentazione software di una Thing a partire dalla sua Thing Description. Viene quindi generata l'interfaccia che implementa il comportamento della Thing.

Tramite il Servient si può istanziare una Exposed Thing, ovvero un'astrazione della Thing ospitata in locale e accessibile dall'esterno tramite lo stack di rete del Servient, consentendo una completa interazione. In questo caso il Servient agisce da server.

Una Consumed Thing, al contrario, è istanziata quando il Servient agisce come client e, partendo dalla TD di una Thing ospitata in remoto esso è in grado di leggerne i metadati e attivare le sue Interaction Affordances, rendendole accessibili al Consumatore.

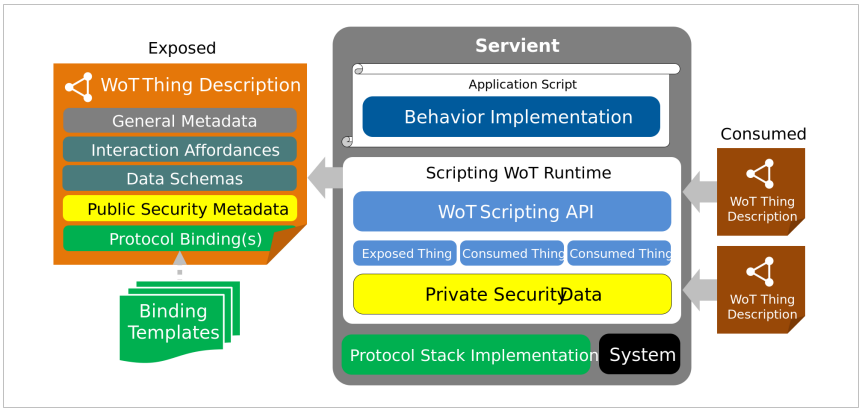


Figura 2.5: WoT Servient [43]

2.3.3 Mashup Application

Una applicazione di mashup è un software in grado di ottenere dati da diverse fonti ed elaborarle fornendo nuove informazioni più complesse e di maggiore utilità [35]. Nel mondo IoT e WoT lo sviluppo di mashup applications è molto comune, poiché spesso le Things sono utilizzate per studi comparativi o statistici, in cui il dato singolo fornito da ogni dispositivo risulta pressochè inutile.
Grazie al Web of Things è molto più semplice implementare applicazioni di mashup, la cui prerogativa intrinseca è quella dell'interoperabilità tra devices.

Un esempio in contesto Smart Home potrebbe essere un'applicazione che tramite una serie di termometri sparsi all'interno dell'abitazione ne calcola la temperatura media e attiva il riscaldamento o l'aria condizionata in base al suo valore.
In contesto industriale si potrebbe pensare invece a un'applicazione che una volta registrata un'anomalia all'interno di una macchina automatica, la segnala direttamente alla casa produttrice e intanto ferma il processo produttivo per permettere a chi di dovere di effettuare le verifiche necessarie.

2.4 Intelligenza Artificiale Generativa

Negli ultimi anni, l'intelligenza artificiale generativa (IAG) ha assunto un ruolo sempre più significativo e trasformativo in molteplici settori, definendo così l'attuale periodo come un'epoca caratterizzata dalla sua influenza. Questa tecnologia, che si basa su algoritmi avanzati di apprendimento automatico, ha rivoluzionato il modo in cui l'uomo crea, comprende e interagisce con l'informazione e l'arte stessa.

Uno dei contributi più rilevanti dell'IAG si ha nell'ambito della creatività computazionale. Attraverso modelli come Generative Adversarial Networks (GAN) e le reti neurali ricorrenti, l'IA può generare immagini, musica, testi e altro ancora, spesso indistinguibili da quelli prodotti da esseri umani. Questo ha portato a una rinascita dell'arte generativa, con opere che sfidano i confini tra l'artificiale e l'autentico.

Parallelamente, l'IAG sta rivoluzionando settori come il design, l'architettura e la moda, consentendo ai professionisti di esplorare un'ampia gamma di possibilità creative in tempi ridotti e con costi contenuti. Nell'ambito informatico, l'IA Generativa sta anche rivoluzionando il processo di sviluppo del software, con algoritmi capaci di generare codice autonomamente, accelerando lo sviluppo e migliorando l'efficienza delle applicazioni.

Tuttavia, l'adozione diffusa dell'IA Generativa non è priva di sfide. Questioni etiche legate alla proprietà intellettuale, alla privacy dei dati e all'equità nell'accesso all'innovazione devono essere affrontate con attenzione.

Inoltre, sorgono interrogativi riguardo alla perdita di posti di lavoro tradizionali e alla necessità di garantire una governance responsabile delle tecnologie emergenti. Nonostante queste sfide, l'Intelligenza Artificiale Generativa continua a definire il nostro periodo storico, offrendo opportunità per l'innovazione e la creatività in una vasta gamma di settori. La sua crescita e il suo impatto sono destinati a persistere e a plasmare il futuro in modi al momento solo parzialmente immaginabili.

2.4.1 Modelli Generativi

La tecnica più utilizzata ad oggi per l'addestramento di un'intelligenza artificiale generativa é quella delle *Generative Adversarial Networks* (GANs) [21]. Esse consistono in una coppia di reti neurali, una definita come *generator* e l'altra come *discriminator*. La prima si occupa di generare il contenuto richiesto (testo, immagine, audio, ...), mentre la seconda, addestrata per la classificazione, di valutarne la qualità, ritornando un valore che esprime la probabilità che il contenuto fornito dal generator sia autentico [15]. Questa tecnica si configura quindi come un gioco in cui il generator deve ingannare il discriminator, facendogli credere che il contenuto prodotto non sia artificialmente generato. Entrambe le reti diventano sempre più precise nel proprio compito, poiché vengono addestrate in maniera alternata tramite *backpropagation*; l'apprendimento può essere quindi visto come l'ottimizzazione di un gioco MiniMax tra due giocatori (generatore e discriminatore) [26]:

$$\min_G \max_D \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

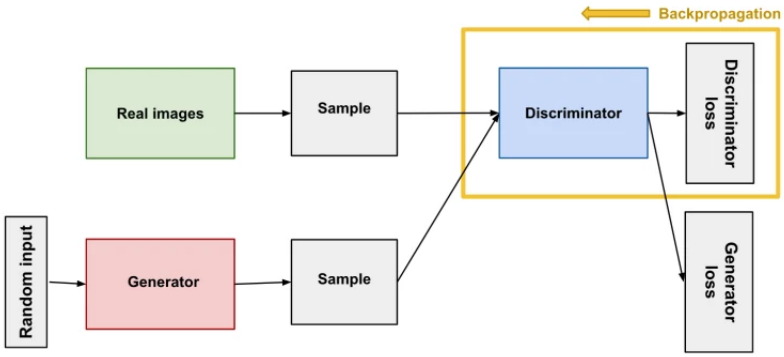


Figura 2.6: Generative Adversarial Network [16]

Un'altra classe di modelli che è spesso utilizzata nell'Intelligenza Artificiale Generativa è quella dei *Transformers*. Sebbene siano noti per le ottime prestazioni nell'ambito del Natural Language Processing (NLP), sono diventati sempre più popolari anche per la generazione di immagini e altri tipi di

dati.
Il loro funzionamento si basa sul concetto di *attenzione* (vengono infatti introdotti per la prima volta nel paper "Attention is All You Need" [40] nel 2017.), ovvero l'abilità di concentrarsi su diverse parti dell'input a seconda di quello che viene richiesto dal problema posto.

L'architettura dei transformers è composta da più *multi-head self-attention layers* e *feed-forward neural networks*. I multi-head self-attention layers consentono al modello di calcolare diverse rappresentazioni di attenzione per ciascun elemento dell'input, consentendo di catturare relazioni semantiche complesse. Le feed-forward neural networks servono a combinare le informazioni ottenute dagli attention layers per generare output utili per compiti specifici, come la traduzione automatica o l'analisi del linguaggio naturale [8].

ChatGPT, sviluppato da OpenAI è uno degli esempi più conosciuti di transformer (in particolare di Generative Pre-trained Transformer), ma sono molti altri i modelli di AI Generativa disponibili oggi e assumono funzionalità diverse [17]. Esistono modelli allenati per convertire testo in immagini (*text-to-image*), e viceversa (*image-to-text*), per convertire testo in video (*text-to-video*), testo in audio (*text-to-audio*), testo in codice (*text-to-code*), oltre che per fornire a una richiesta testuale una risposta testuale (*text-to-text*). Di seguito alcuni esempi:

- *text-to-image*: DALL-E 3, Stable Diffusion, Imagen, Midjourney;
- *image-to-text*: GPT4-Vision, Flamingo;
- *text-to-video*: Sora, Phenaki;
- *text-to-audio*: AudioLM, JukeBox;
- *text-to-code*: Codex, AlphaCode, Copilot, CodeWhisperer, Code Llama;
- *text-to-text*: ChatGPT-4, LaMDA, Bard, PEER, OpenChat.

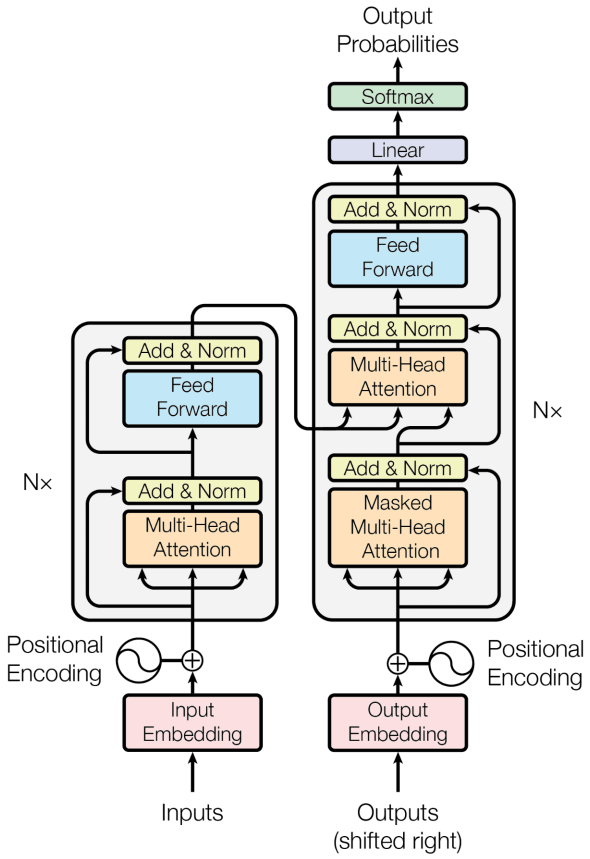


Figura 2.7: Transformer [40]

La maggior parte di questi modelli sono implementati da BigTechs come OpenAI, Google, Meta, DeepMind, Nvidia.

2.4.2 Generazione di Testo e Codice

La generazione di testo e codice tramite modelli generativi basati sull'intelligenza artificiale è diventata sempre più popolare e potente negli ultimi anni, grazie a sviluppi significativi nel campo del deep learning, in particolare con modelli come i Generative Pre-trained Transformers.

Questi modelli sono in grado di generare testo e codice in modo coerente e realistico, a volte quasi indistinguibile da quello umano [10]. Utilizzano

l'architettura transformer, che permette loro di catturare relazioni complesse nei dati di input e di produrre output coerenti e di alta qualità.

Oltre alla generazione di testo, che è sicuramente una funzionalità più trasversalmente utile, in quanto può essere usata in diversi contesti, anche la generazione di codice negli ultimi tempi sta attirando l'attenzione della ricerca e delle industrie [11]. Sono infatti diversi i campi in cui può essere applicata, risultando in una riduzione del tempo utilizzato per eseguire compiti ripetitivi e non particolarmente complessi, ad esempio per il refactoring di codice, o per la correzione di errori durante il processo di sviluppo.

I benefici che offre l'integrazione dell'intelligenza artificiale nella fase di scrittura del codice sono diversi ed a trarne vantaggio non sono solo gli sviluppatori che possono dedicarsi a compiti più complessi, risparmiando tempo, ma di conseguenza anche le aziende che traducono questo risparmio di tempo in risparmio economico e infine il cliente finale che riceve il prodotto in minor tempo e con una qualità maggiore, dato il controllo ulteriore che viene fatto dall'intelligenza artificiale sul codice [38].

Non mancano però risvolti negativi nell'uso dell'intelligenza artificiale per la generazione di codice. È infatti una tecnologia ancora in via di sviluppo e la sua precisione non risulta elevata [19]: ciò al momento non la rende utilizzabile in tutti i contesti. In futuro la sua integrazione più diffusa potrebbe portare alla riduzione di posti di lavoro, soprattutto per quanto riguarda i *junior-programmers*, non ancora in grado di occuparsi di incarichi complessi, oltre ad ampliare il divario tra aziende (o addirittura nazioni) con diverse possibilità di investimento.

Inoltre per permetterne un utilizzo più ampio è necessario elaborare una regolamentazione ed educare i possibili fruitori a farne un uso etico e responsabile.

Capitolo 3

WoT Conductor

WoT Conductor è un'applicazione Android Native che si basa sugli standard del W3C Web of Things (WoT) che consentono l'interoperabilità tra dispositivi e servizi nell'Internet of Things (IoT), evitando la frammentazione che caratterizza quest'ultimo. In questo capitolo vengono delineati gli obiettivi dell'applicazione, ne vengono descritti i requisiti tecnici e funzionali e illustrati gli elementi costitutivi.

3.1 Obiettivo

L'obiettivo principale nello sviluppo dell'applicazione è stato quello di arrivare a un prodotto che consentisse all'utente di interagire in maniera semplice e immediata con tutti i dispositivi IoT a sua disposizione.

WoT Conductor ha il compito di gestire, come un direttore d'orchestra (da qui il suo nome), i diversi devices disponibili, partendo da un input dell'utente. Per fare ciò vengono create delle *Routines*, ovvero *mashup applications* che sfruttano azioni e proprietà di una o più Things per automatizzare attività o procedure.

L'utente ha diversi modi di interagire con l'applicazione per creare una Routine. Grazie all'integrazione di un'intelligenza artificiale generativa è possibile generare una nuova Routine tramite un comando testuale o vocale. Per

la descrizione di Routines più elaborate l'utente può servirsi di un'interfaccia a blocchi tramite la quale progettare l'automazione desiderata.

Figura 3.1: Esempio di interazione tramite WoT Conductor

3.2 Requisiti

Di seguito sono definiti i requisiti tecnici e funzionali che l'applicazione deve soddisfare.

- **Connessione a Thing Description Directory**: La connessione ad una TDD è necessaria per ottenere le Thing Descriptions delle Things disponibili al momento sulla rete. La TDD deve essere ospitata sulla rete locale. Può essere *automatica* o *manuale*:
  - *automatica*: tramite un processo di DNS Service Discovery l'applicazione deve essere in grado di identificare e connettersi alla TDD che si pubblicizza sulla rete locale mediante messaggi *multicast DNS*.
  - *manuale*: l'utente deve essere in grado di inserire manualmente l'indirizzo IP corrispondente alla TDD.

- **Visualizzazione delle Things disponibili**: L'utente deve poter visualizzare tutte le Things disponibili in tempo reale, con le relative proprietà osservabili e azioni invocabili.
- **Notificazione all'utente di aggiornamenti**: Aggiornamenti all'interno della TDD devono essere notificati all'utente in tempo reale.
- **Interazione con assistente virtuale**: L'utente deve poter rivolgersi ad un assistente virtuale implementato tramite intelligenza artificiale generativa per descrivere la Routine che vuole creare. Può immettere un comando:
  - *vocale*: tramite il microfono del dispositivo;
  - *testuale*: usando la tastiera del dispositivo.
- **Utilizzo di interfaccia a blocchi**: Deve essere garantito l'accesso a un'interfaccia a blocchi che consenta all'utente di creare una Routine componendo un flusso di blocchi tramite UI.
- **Visualizzazione Routines**: È necessario che tutte le Routines create vengano mostrate in una apposita sezione, e che vengano distinte quelle eseguibili nel momento della visualizzazione, da quelle non eseguibili.
- **Esecuzione di una Routine**: Ogni Routine deve poter essere eseguita sul dispositivo in cui è installata l'applicazione oppure sull'*Edge*. L'utente deve poter scegliere dove eseguire le proprie Routines.
- **Schedulazione di una Routine**: L'utente deve poter impostare un orario nel futuro in cui eseguire una Routine.
- **Eliminazione di una Routine**: L'utente deve poter eliminare ogni Routine dalla visualizzazione.
- **Rinominazione di una Routine**: L'utente deve poter rinominare ogni Routine.

- **Visualizzazione del codice di una Routine**: L'utente deve essere in grado di visualizzare il codice implementativo di ogni Routine.
- **Scelta dei Binding Templates**: L'utente deve avere la possibilità di scegliere, tra quelli implementati, quali Binding Templates utilizzare.

3.3 Architettura

Per funzionare in maniera corretta WoT Conductor deve essere utilizzata in abbinamento a un modulo secondario, ospitato sull'Edge, chiamato *WoT Flow*. Questa componente continene infatti una Thing Description Directory, l'implementazione di un'interfaccia a blocchi per la creazione delle Routines e il server che si occupa della gestione delle Routines.

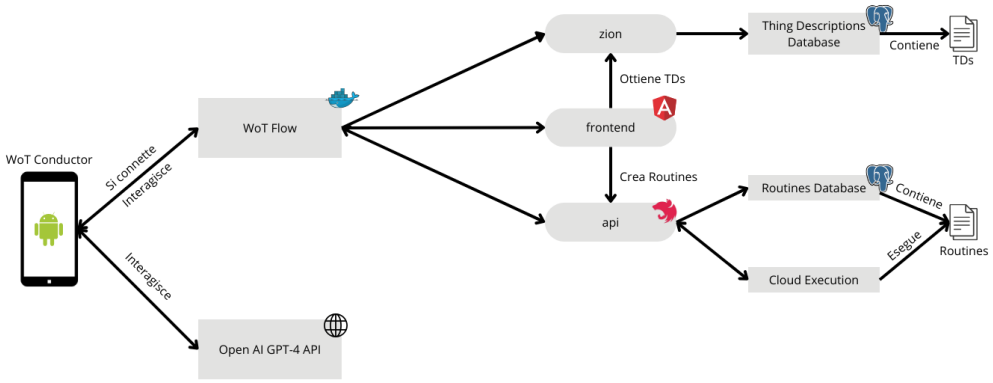


Figura 3.2: Struttura WoT Conductor

3.3.1 Applicazione Android

L'applicazione WoT Conductor fornisce all'utente una GUI per interagire con tutto il sistema sottostante. Include come modulo secondario un WoT

Servient. Le sue funzionalità principali sono la Discovery e consumazione di Things e la Creazione e gestione di Routines.

Discovery e consumazione di Things

Per poter interagire con le Things è necessario ottenerne le Thing Descriptions per poi procedere, tramite Servient alla consumazione di esse. Questo processo è uno dei processi chiave dell'applicazione. All'avvio, infatti, l'applicazione cerca di connettersi, automaticamente o tramite l'indirizzo IP fornito dall'utente, alla TDD ospitata nel modulo WoT Flow sulla rete locale per ottenere tutte le TDs contenute al suo interno. Una volta acquisite, tramite il WoT Servient vengono da esse generati degli oggetti *Consumed Things* che permettono l'utilizzo delle Interaction Affordances di ogni Thing.

L'utente può visualizzare tutte le Things disponibili e le relative proprietà e azioni all'interno di un'apposita schermata che è costantemente aggiornata. L'applicazione, infatti, una volta connessa alla TDD e ottenute tutte le Thing Descriptions contenute al suo interno, rimane in ascolto per poter segnalare all'utente tramite notifiche e Toasts l'aggiunta, la rimozione e la modifica di TDs.

Creazione e gestione di Routines

Una schermata di WoT Conductor è interamente dedicata alla creazione di Routines. L'utente può scegliere se creare la Routine in maniera automatica servendosi dell'assistente virtuale (GPT-4) tramite input vocale o testuale, o se generarla attraverso la composizione di un diagramma di flusso a blocchi mediante l'editor flow-based fornito dal modulo WoT Flow.

Una volta creata, la Routine viene resa disponibile all'interazione in un'altra schermata, quella per la gestione delle Routines. Qui l'utente può visualizzare tutte le Routines create nel tempo e, tramite una serie di bottoni, interagire con esse. Sono infatti disponibili diverse funzionalità: rinominazione, eliminazione, consultazione di codice, esecuzione e schedulazione.

É possibile visualizzare i risultati ritornati dalle Routines in apposite schermate, apribili cliccando sulle snackbars che vengono generate una volta che l'esecuzione giunge al termine.

3.3.2 Modulo WoT Flow

Il modulo WoT Flow si divide in tre parti: TDD Zion, Frontend e API.

TDD Zion

Zion è un'implementazione di Thing Description Directory. Contiene le Thing Descriptions dei dispositivi dell'utente, seguendo gli standard WoT e le fornisce all'applicazione Android.

Frontend

La componente *Frontend* del modulo WoT Flow consiste in un'interfaccia tramite la quale, in base alle TDs disponibili all'interno della TDD, l'utente può combinare diverse azioni e proprietà di una o più Things per creare a proprio piacimento automazioni, tramite una visualizzazione a blocchi flow-based. Una volta composta la Routine, ad essa viene assegnato un nome e viene resa disponibile all'utente nella schermata di gestione delle Routines dell'app Android.

API

API é la componente backend di WoT Flow che gestisce il salvataggio delle Routines e tutte le azioni correlate. Permette inoltre l'esecuzione delle Routines in Edge, anziché direttamente sul dispositivo Android.

Capitolo 4

Implementazione

Durante la realizzazione del progetto è stato necessario utilizzare diverse componenti e tecnologie, a causa dell'eterogeneità dei requisiti. Di seguito sono illustrate la struttura del codice e le scelte implementative, con descrizioni dettagliate per ogni componente.

Il codice dell'applicazione Android *WoT Conductor* e del modulo *WoT Flow* è open-source e accessibile al seguente link `https://github.com/UniBO-PRISMLab/wot-conductor`.

4.1 Struttura e dettagli del codice

WoT Conductor è un'applicazione costituita da due moduli: il modulo *app* e il modulo *wot-servient* a loro volta composti da diversi *packages*.

WoT Flow è un'applicazione web che include tre diverse componenti: la TDD Zion, la cui implementazione è open-source (e consultabile direttamente tramite questo link `https://github.com/vaimee/zion`), la componente *frontend*, implementata in *AngularJS* e costituita di diversi *components*, e la componente *api* - un server NestJS formato da due moduli (*routine* e *execute*). Per una più semplice integrazione e un environment più solido, le tre componenti sono *dockerizzate* tramite l'uso di *docker-compose*.

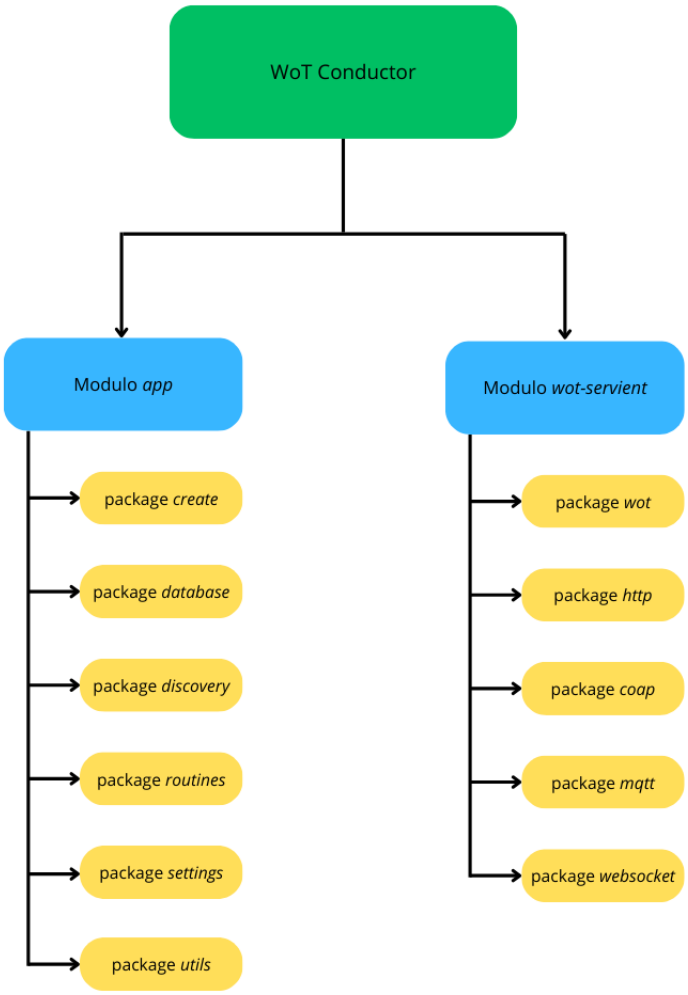


Figura 4.1: Struttura del codice di WoT Conductor

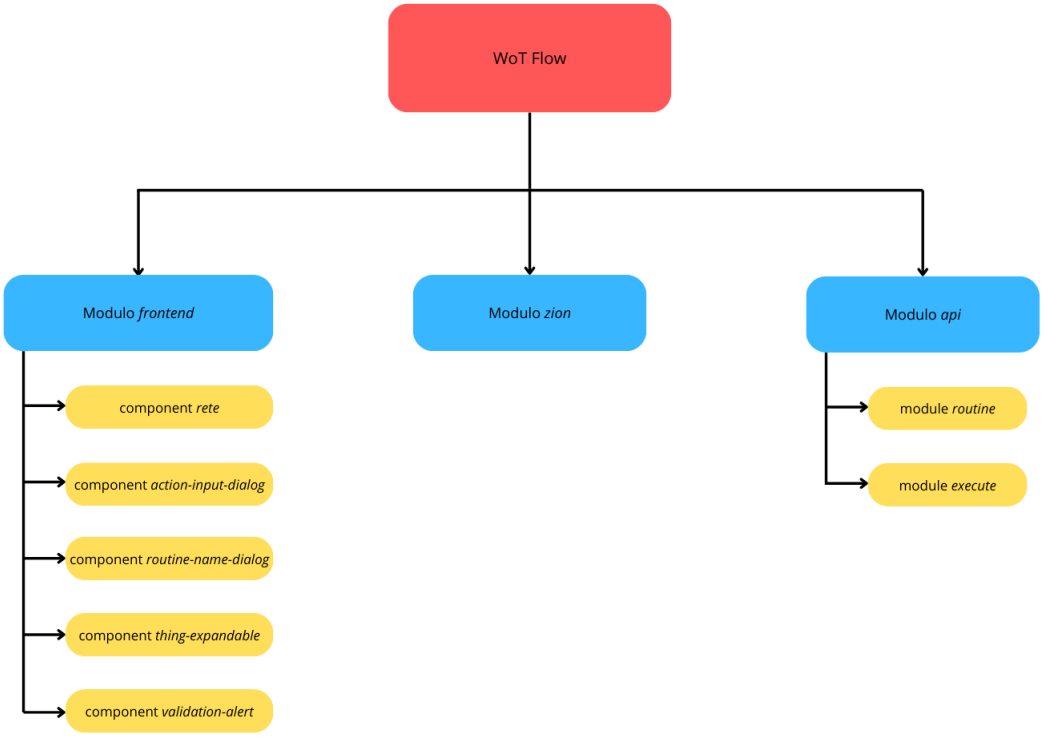


Figura 4.2: Struttura del codice di WoT Flow

4.1.1 WoT Conductor - modulo *wot-servient*

Il modulo *wot-servient*, fa parte del progetto *AndroidWotServient*[1] e contiene cinque *packages*. Il package *wot* si occupa della gestione delle funzionalità di base che un Servient WoT deve fornire. I restanti quattro packages implementano quattro Binding Templates: *HTTP, CoAP, MQTT* e *WebSocket*.

Rispetto all'implementazione originale sono state aggiornate le dipendenze e sono stati aggiunti dei metodi per l'ottenimento di properties e actions all'interno delle classi `Thing` e `ConsumedThing`

[1] `https://amslaurea.unibo.it/24213/`

4.1.2 WoT Conductor - modulo *app*

Il modulo *app* si compone di sei *packages*. Quattro di essi corrispondono alle quattro schermate che compongono l'interfaccia dell'applicazione: *discovery, routines, create* e *settings*. Il package *utils* implementa delle classi e dei metodi "helpers" per l'esecuzione di specifici tasks, mentre il package *database* si occupa della gestione del database dell'applicazione.

Main Activity

La *Main Activity* rappresenta la prima schermata che appare all'utente all'apertura dell'applicazione. In Wot Conductor la Main Activity ospita una *BottomNavigationView* e una *FragmentContainerView* che abbinate all'Android Jetpack's Navigation framework consentono di sviluppare un servizio di Navigation fluido e responsive.

La *BottomNavigationView* corrisponde alla barra di navigazione che si trova nella parte inferirore dello schermo, tramite la quale l'utente può scegliere quale schermata visualizzare. Il numero di "selettori di schermata" e le loro caratteristiche (titolo, icona, id) sono definiti in un apposito menu file: *bottom_navigation_menu.xml*.

La *FragmentContainerView* nel contesto del Navigation framework assume la funzione di *NavHostFragment*, ossia un container all'interno del quale si alternano diversi Fragments (ciascuno dei quali è associato a un item del *bottom navigation menu*). Tutti i possibili passaggi da un Fragment ad un altro sono descritti tramite un Navigation Graph: una risorsa XML che connette tutte le destinazioni (Fragments) tramite azioni (eventi). L'alternarsi dei Fragments è regolato dal *NavigationController*, fulcro del Navigation framework.

Una volta impostati questi elementi viene creato il binding tra *BottomNavigationView* e *NavigationController*, in modo da associare ad ogni item del *bottom navigation menu* un'azione del *NavigationGraph* controllata dal *NavigationController*.

Dalla Main Activity viene avviato il processo di Thing Discovery, che rimane attivo finchè l'applicazione non viene terminata. Esso è gestito dalle classi implementate nel pacchetto *discovery*.

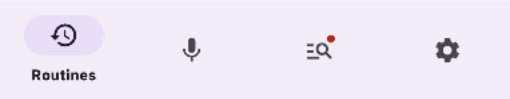


Figura 4.3: BottomNavigationView

Package *discovery*

Il package *discovery* è responsabile del funzionamento del processo di *Thing Discovery*. Esso può essere automatico o manuale: nel primo caso viene attivata, tramite la classe `NSDUtils` del pacchetto *utils*, la DNS Service Discovery della TDD Zion sulla rete locale, mentre nel secondo caso l'utente tramite le impostazioni inserisce direttamente l'indirizzo IP corrispondente alla Thing Description Directory.
Una volta raggiunta la TDD, mediante la classe `TDDExplorer` si procede all'ottenimento e alla consumazione delle TDs.

La classe `TDDExplorer` implementa l'interfaccia `Runnable`, in modo da poter eseguire il suo metodo `run()`, in un Worker Thread, evitando così di sovraccaricare il thread UI (o Main Thread) e bloccare l'interfaccia grafica dell'applicazione. All'interno del metodo `run()` viene stabilita la connessione HTTP alla TDD, mediante l'utilizzo della classe `HTTPUrlConnection`, parte del package `java.net`, che permette la gestione di una connessione tramite URL con supporto per le features HTTP. Stabilita la connessione, si procede quindi all'invocazione di una `GET` request in modo da ottenere in risposta tutte le TDs contenute all'interno della TDD. Quindi, la risposta viene elaborata, e, da un `JSONArray` si passa ad un oggetto `Thing`, mediante l'uso di un metodo appartenente al modulo *wot-servient*. La `Thing` viene subito "consumata" e salvata all'interno del database dell'applicazione, così da essere disponibile universalmente.

Il processo appena descritto viene avviato dalla Main Activity su un thread secondario, che comunica con essa stessa tramite un sistema di callbacks, in modo da tenere aggiornato l'utente sullo stato della Discovery: se durante la connessione alla TDD dovesse essere lanciata qualche eccezione, all'utente viene segnalato l'errore tramite un *Toast*. Se la Discovery va a buon fine, tramite la callback, viene ritornata alla Main Activity un'`ArrayList` contenente tutte le `ConsumedThing`s ottenute e viene avviato l'aggiornamento in tempo reale delle Things disponibili.

Per fare in modo che non vengano eseguite Routines in cui sono coinvolte Things che al momento dell'invocazione non sono disponibili è necessario che il database dell'applicazione sia costantemente aggiornato. L'API della TDD Zion usa il protocollo SSE (*Server Sent Events*), tramite il quale il server può comunicare con il client per mezzo di messaggi, considerati eventi, che contengono dati. In questo caso il server è proprio la TDD Zion e il client è WoT Conductor.
Attraverso una libreria esterna è possibile fare si che l'applicazione si metta in ascolto dei Server Sent Events.

Gli eventi che la TDD è in grado di generare sono: `"thing_created"`, `"thing_updated"`, `"thing_deleted"`. Sono gestiti da un `SSEListener` nella Main Activity (poichè deve rimanere in ascolto durante tutto il run-time dell'applicazione), che, oltre ad aggiornare il database, ogni volta che riceve un evento, notifica l'utente con un badge sulla BottomNavigationView e con un toast, se l'utente sta visualizzando la schermata *"Discover"*, con una notifica (gestita dalla classe `NotificationUtils`), altrimenti.

Le Things ottenute dal processo di Thing Discovery sono presentate all'utente mediante una lista di blocchi espandibili. Inizialmente è visualizzabile solo il nome di ciascuna Thing; con un tocco si apre una tendina in cui sono elencate le azioni e le proprietà corrispondenti alla Thing stessa. Tutto ciò è implementato all'interno del *DiscoverFragment*. La lista è realizzata tramite l'utilizzo di una `RecyclerView` di `Expandable`s, cards espandibili composte di *header* (ciò che viene mostrato quando l'`Expandable` è chiuso) e *content*

(ciò che viene rivelato al tocco), sviluppate nella libreria *Material Expansion Panel*.

La `RecyclerView` viene aggiornata in maniera dinamica per mezzo di un `ViewModel`, una componente che permette di gestire e immagazzinare dati UI-related compatibilmente con il ciclo di vita dell'applicazione, rendendoli più consistenti. Il `ViewModel`, offre una serie di metodi *getter*, tra i quali uno che permette di ottenere tutte le (`Consumed`)`Things` disponibili all'interno del database, attraverso un oggetto `LiveData`. La classe `LiveData` permette di istanziare oggetti "*osservabili*" che coerentemente allo stato della lifecycle della componente in cui si trovano (i.e. Activity, Fragment), notificano gli "iscritti" quando si verifica un cambiamento nei dati osservati. Appena viene rilevato un cambiamento all'interno del database (aggiunta, rimozione o modifica di una Thing), quindi, la `RecyclerView` viene aggiornata in automatico, tramite l'uso della *utility-class* `DiscoverDiffUtil`, che calcola la differenza tra due liste (`old` e `new`) e notifica al `RecyclerViewAdapter` le modifiche che deve applicare per presentare correttamente i dati aggiornati.

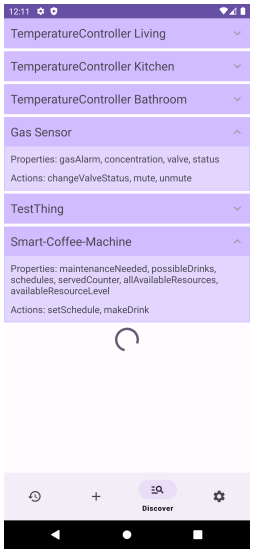


Figura 4.4: Schermata Discover

Package *create*

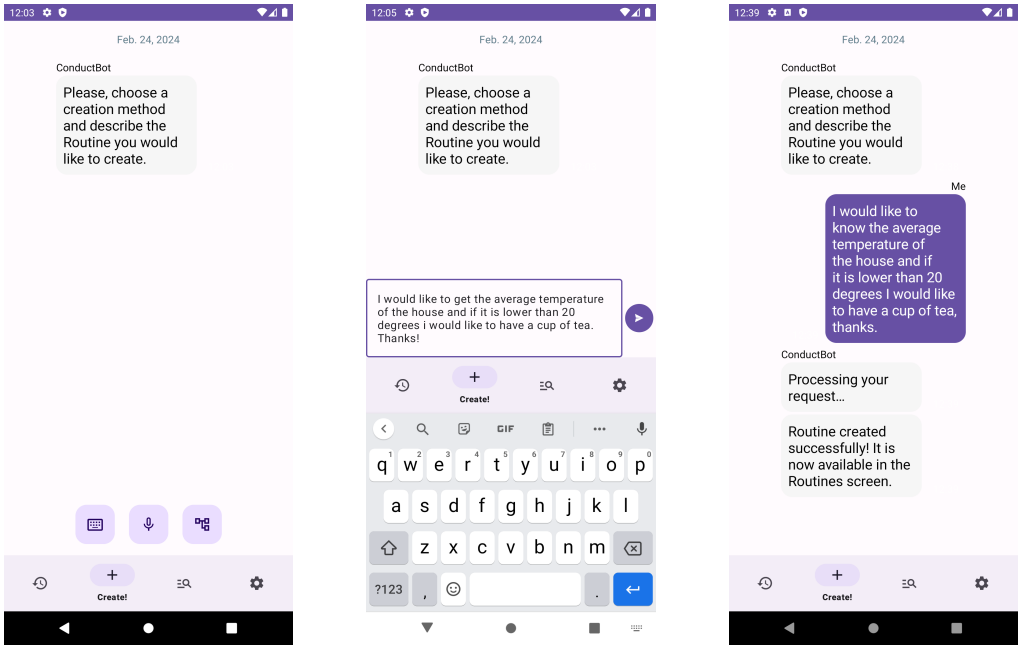
Il pacchetto *create* contiene l'implementazione dei metodi di creazione di una Routine. L'utente può scegliere se servirsi dell'aiuto dell'assistente virtuale implementato mediante l'utilizzo di IA Generativa (*OpenAI GPT-4*), o se utilizzare l'interfaccia grafica a blocchi.

La creazione di una Routine consiste nella generazione di codice (Java, per l'esecuzione sul dispositivo Android; TypeScript per l'esecuzione sull'Edge) e nell'assegnazione di un nome alla Routine stessa.

- **Assistente virtuale**: l'utente interagisce con l'assistente virtuale per mezzo di una Chat UI, con un comando vocale (viene usato un `RecognizerIntent`) o testuale (tramite un `EditText`). Viene quindi istanziato un oggetto `OpenAIUtils`, che contiene le informazioni necessarie per l'accesso alle API OpenAI e tramite il suo metodo `sendGPT()` viene creato il body della richiesta per l'API, composto da:
  - `System Content`: una stringa di testo che istruisce il GPT, fornendo un contesto, delle regole e un esempio di codice (Java o TypeScript, in base a dove l'utente sceglie di eseguire la Routine) che ne permettono il tuning. Viene anche specificato il formato (JSON) e la struttura che deve avere la risposta fornita. La stringa è sempre la medesima.
  - `User Content`: composto dal comando inserito dall'utente e dalla lista delle TDs disponibili all'interno della TDD nel momento della richiesta, per fornire all'Intelligenza Artificiale l'accesso a tutte le informazioni necessarie alla scrittura di codice per l'esecuzione della Routine da creare.

  Una volta ricevuta, la risposta (in formato JSON e strutturata come specificato nel *System Content*) viene trasformata tramite il metodo `processResponse` in un oggetto `OpenAIResponse`. Gli attributi dell'oggetto vengono poi utilizzati per creare, tramite una POST request al server WoT Flow/*api*, la nuova Routine.

- **Interfaccia a blocchi**: è implementata dal modulo WoT Flow/*frontend*. Viene resa accessibile all'interno dell'applicazione da una `WebView`.



(a) Scelta del metodo di creazione

(b) Inserimento comando testuale

(c) Risposta dell'assistente virtuale

Figura 4.5: Schermata Create

Package *routines*

Il pacchetto *routines* permette la gestione delle automazioni create. Il modulo WoT Flow assume un ruolo centrale nel funzionamento di questo package siccome ospita il database che contiene tutte le Routines e il server che ne consente l'aggiornamento.

Per un corretto funzionamento dell'applicazione è necessario, che, come viene fatto per le Things, anche le Routines siano salvate all'interno del database locale per consentirne una più facile gestione. Ciò viene fatto in maniera pressoché analoga a quella descritta per la Discovery iniziale delle Things, tramite un processo definito "*Esplorazione*": viene istanziato un oggetto `RoutinesExplorer` che implementa l'interfaccia `Runnable` e

viene eseguito su un worker thread. Durante la sua esecuzione, tramite una `HTTPUrlConnection`, l'API *wot-flow* viene interrogata e restituisce in risposta un `JSONArray` di Routines, che vengono poi trasformate in oggetti Routines, mediante l'utilizzo dell'oggetto `ObjectMapper`, parte della libreria *jackson-databind* che permette la deserializzazione da stringhe JSON a oggetti Java, per mezzo delle *annotations* `@JsonProperty` utilizzate nella classe `Routine`. Gli oggetti `Routine` vengono quindi salvati nel database dell'applicazione. Viene utilizzato un sistema di callbacks per la comunicazione all'utente di eventuali errori durante il recupero delle informazioni dal server.

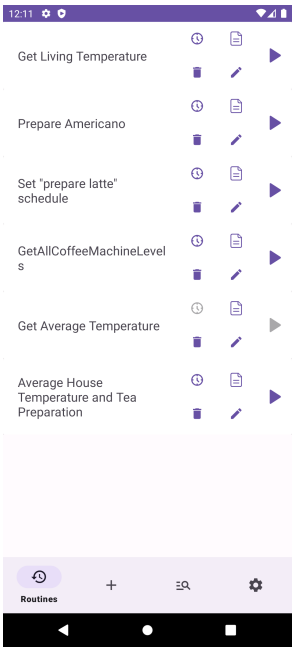


Figura 4.6: Schermata Routines

A differenza della TDD Zion, il server *wot-flow* non supporta i SSE, quindi, per mantenere la lista di Routines sempre aggiornata, il processo di recupero dal server (che include una pulizia totale della tabella Routines all'interno del database locale) viene eseguito ogni volta che il `RoutinesFragment` è "`RESUMED`".

L'utente è in grado di interagire con le automazioni disponibili tramite l'interfaccia, in cui sono elencate tutte le Routines, ognuna associata a 5 bottoni che corrispondono a 5 diverse funzioni: Esecuzione, Scheduling, Rimozione, Consultazione del codice e Rinominazione. Tutti i metodi necessari alla gestione di ogni Routine sono definiti all'interno della classe `RoutinesViewHolder` che si occupa di assegnare ad ogni bottone una funzionalità specifica.

Figura 4.7: Bottoni per la gestione delle Routines

- **Esecuzione**: L'esecuzione di una Routine può avvenire interamente sul dispositivo Android, per mezzo del Servient WoT implementato nel modulo *wot-servient* oppure sull'Edge mediante la funzionalità `execute` fornita dal modulo WoT Flow/*api*.

  Per quanto riguarda l'esecuzione sul device Android, visto che il codice di ogni Routine viene ottenuto a run-time è necessario l'utilizzo di uno strumento che permetta di eseguirlo in maniera dinamica: *beanshell*. L'esecuzione dinamica di codice è indubbiamente rischiosa dal punto di vista della sicurezza dell'applicazione, aspetto che è stato volutamente trascurato durante lo sviluppo di questa parte.

  La possibilità di eseguire una Routine dipende dalla disponibilità di tutte le Things coinvolte al momento dell'esecuzione. La classe Routine, infatti, implementa l'attributo `thingIds`: una stringa, contenente un `JSONArray` di tutti gli identificativi delle Things che compongono una Routine. Il metodo `checkThingsAvailability()` abilita e disabilita

il bottone di esecuzione e quello di scheduling di una Routine in base alla disponibilità delle Things.

Quando viene premuto il bottone di esecuzione (se abilitato), le `ConsumedThings` necessarie vengono ottenute dal database locale e inserite all'interno di una `Map`, che viene passata all'interprete *beanshell*, insieme all'istanza in cui viene eseguita la Routine. Oltre alla possibilità di passare oggetti e variabili all'interprete, la libreria *beanshell* consente di ottenere, tramite il metodo `get()` variabili e oggetti creati durante il runtime, come ad esempio la lista di *properties* osservate, che vengono mostrati all'utente in una Activity dedicata (`ResultsActivity`) accessibile tramite la *snackbar* che appare nella schermata una volta che l'esecuzione giunge al termine.
Tutto ciò è eseguito all'interno di un thread secondario fornito dalla classe `Executors`.

- **Scheduling**: Lo *scheduling* non è altro che un'esecuzione in un momento preciso stabilito dall'utente.
  La logica dell'esecuzione è analoga a quella descritta precedentemente.

  L'utente sceglie l'orario in cui eseguire la Routine attraverso un `TimePickerDialog`. Se l'orario scelto corrisponde a un momento già passato nella giornata che si sta svolgendo, allora la Routine viene eseguita il giorno successivo all'ora indicata.

  La schedulazione è gestita da un `AlarmManager`, oggetto incaricato di avviare un Intent in un momento preciso del futuro, anche quando l'applicazione è chiusa. Un `BroadcastReceiver`, nominato `AlarmReceiver` rimane in ascolto e una volta ricevuto l'Intent inviato dall'`AlarmManager` fa partire l'esecuzione della routine.

- **Rimozione**: La rimozione di una Routine viene gestita dal metodo `deleteRoutine()` che in primo luogo rimuove dal database del server *wot-flow* la Routine scelta, tramite un metodo definito nella classe `RoutineAPIQueries` in cui viene utilizzato un `OkHttpClient` (client

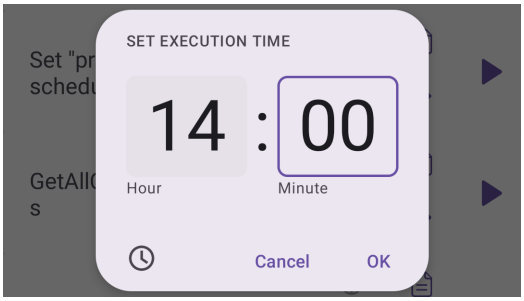


Figura 4.8: Dialog per lo scheduling di una Routine

della libreria *okhttp*) per eseguire una `DELETE` query. In secondo luogo la Routine viene rimossa anche dal database locale.

- **Rinominazione**: Quando l'utente tocca il bottone per la rinominazione di una Routine, gli viene presentato un `Dialog`, contenente un `TextInputEditText`, in cui può essere immesso il nuovo nome da assegnare alla Routine. Se il nuovo nome corrisponde a una stringa vuota viene generato un errore e restituito all'utente tramite un Toast.
  L'aggiornamento della Routine, come la rimozione consiste in una query all'API del server *wot-flow* tramite un metodo della classe `RoutineAPIQueries` e l'aggiornamento della Routine in questione all'interno del database.

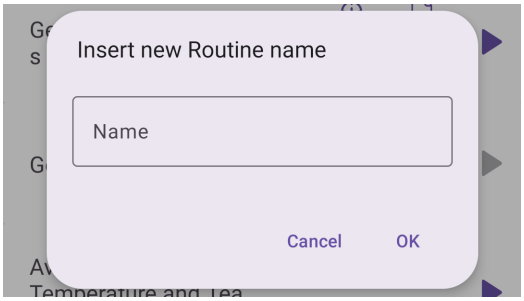


Figura 4.9: Dialog per la rinominazione di una Routine

- **Consultazione codice**: Attraverso il bottone di consultazione del codice, l'utente è in grado di visualizzare il codice generato durante fase di creazione. Il codice di ogni Routine è memorizzato all'interno degli attributi `javaCode` e `tsCode`.

Quando l'utente preme il bottone, viene avviata una nuova Activity (`RoutineCodeActivity`) a cui viene passato l'attributo `javaCode` o `tsCode`, tramite un Intent. Nella `RoutineCodeActivity` il codice viene processato da un *syntax highlighter* che ne rende la lettura più immediata.

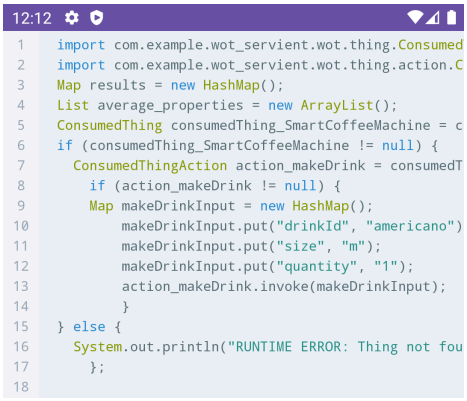


Figura 4.10: Codice di una Routine

L'interfaccia della sezione Routines é gestita dal `RoutinesFragment`. Le Routines sono presentate all'utente tramite una `RecyclerView`, composta di `Cards`, contenenti il nome della Routine e i 5 bottoni che ne permettono la gestione. L'aggiornamento della `RecyclerView` funziona in maniera totalmente analoga a quello del `DiscoverFragment`, mediante l'uso di `ViewModel`, `LiveData` e `DiffUtil`.

Package *settings*

Il pacchetto *settings* contiene il `SettingsFragment` che mediante la libreria *AndroidX Preference* implementa la schermata impostazioni dell'applicazione. La classe `Settings` contiene invece dei metodi helpers per una più semplice gestione e interazione con le *SharedPreferences*. Il metodo `getIP()` ottiene dalle impostazioni l'indirizzo IP trovato tramite DNS Service Discovery o inserito manualmente dall'utente. I metodi `getTDDUrl()`, `getFlowApiUrl()` e `getFlowFrontUrl()`, utilizzano il metodo `getIP()` per

comporre gli Url corrispondenti ai moduli zion, frontend e api di WoT Flow. Il metodo `executeOnEdge()`, invece, ritorna un valore booleano corrispondente alla scelta fatta dall'utente in merito all'esecuzione delle Routines sull'Edge.

Dalle impostazioni l'utente può quindi scegliere le modalità di discovery delle Things e di esecuzione delle Routines. Inoltre può configurare i Protocol Bindings a suo piacimento.

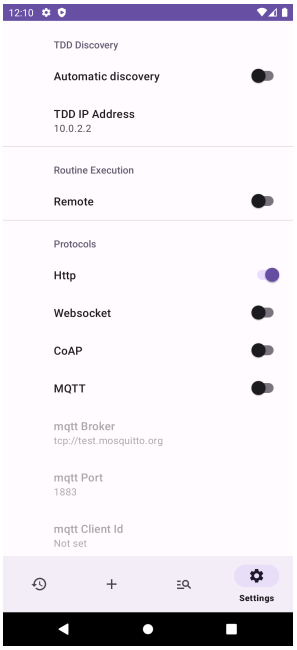


Figura 4.11: Schermata Impostazioni

Package *database*

L'accesso al database interno (viene utilizzato un database *Room*) è regolato dalla classe `WotInteraction`, in cui viene creata un'istanza dello stesso, sulla quale vengono eseguite le query definite nei file `DAO`, del package *database*.
Tutte le queries vengono eseguite su un worker thread, per evitare che la UI dell'applicazione si blocchi.

Oltre ai file DAO, all'interno del package *database* sono contenute anche le implementazioni di ogni *Entity*: `EntityConsumedThings` e `EntityRoutines`. Ad ogni entità del database è abbinato un *Repository*, ovvero una componente che funge da intermediario tra il `ViewModel` e il data source (in questo caso il db *Room*), permettendo così una maggiore astrazione e separazione tra *business logic* e User Interface.

Package *utils*

Il pacchetto *utils* contiene diverse classi:

- `CodeFormatterUtils`: implementa una serie di metodi per la formattazione e regolazione dei livelli di indentazione di una stringa di codice Java;

- `NotificationUtils`: al suo interno viene creato un `NotificationChannel` con `IMPORTANCE_DEFAULT` sul quale indirizzare tutte le notifiche dell'applicazione.
  Inoltre vi sono definiti i metodi responsabili della creazione delle notifiche per ogni evento riguardante l'aggiornamento in tempo reale delle Things disponibili. In generale prima che una notifica venga creata si verifica che l'applicazione abbia il permesso di lanciarla, dopodiché la essa viene "buildata" tramite il `NotificationCompat.Builder` e mostrata all'utente. Ad ogni tipo di evento è associato un ID diverso, così da non creare una nuova notifica ogni volta, aggiornando la notifica precedente caratterizzata dallo stesso ID. Questo permette una gestione delle risorse e una UX migliori.

- `NSDUtils`: si occupa della DNS Service Discovery, un processo che consiste nel domandare a tutti i dispositivi connessi sulla rete locale (tramite messaggi *multicast-DNS*) a chi corrisponda un determinato `SERVICE_TYPE`. In questa classe sono implementati metodi per l'inizializzazione di `DiscoveryListener` e `ResolveListener`, oltre che metodi per l'avvio della discovery stessa. Il `SERVICE_TYPE` utilizzato dalla

TDD Zion per pubblicizzarsi è `._wot._tcp`, ed è quindi il `SERVICE_TYPE` impostato come filtro per la ricerca. Una volta trovato il servizio, si procede alla risoluzione dell'indirizzo IP che viene passato, attraverso la `NSDDiscoveryCallback` alla MainActivity la quale si occupa di rendere quest'informazione disponibile all'utente tramite le *SharedPreferences*.

4.1.3 WoT Flow - modulo *zion*

Il modulo *zion* implementa la Thing Description Directory che contiene tutte le TDs corrispondenti alle Things dell'utente. Fornisce una API che consente di ottenere, modificare ed eliminare una o più Thing Descriptions. Le sue funzionalità principali consistono nella pubblicizzazione tramite mDNS del servizio `._wot._tcp` e nell'emissione di eventi ogni volta che avviene una modifica al suo interno. È accessibile dalla porta 3000 del suo docker container.

4.1.4 WoT Flow - modulo *frontend*

Il modulo *frontend* fornisce all'utente un'interfaccia a blocchi flow-based, in stile *NodeRED*, che permette la progettazione grafica di Routines e la generazione automatica del relativo codice Java, partendo dalle Things disponibili all'interno della TDD e dalle relative Interaction Affordances. Consiste in un'applicazione web implementata in AngularJS, che dialoga con il modulo backend *api*. È formato da diversi *components*. Un component è un elemento Angular chiave e consiste in un file HTML che definisce ciò che viene renderizzato, un file TypeScript che ne definisce il comportamento e un selector CSS che definisce come il component debba essere usato in un template. Questo modulo è associato alla porta 4200 del suo docker container.

Component *rete*

Il component *rete* costituisce il nucleo del modulo *frontend*, e basa il suo funzionamento sulla libreria *retejs* che fornisce metodi e interfacce per la

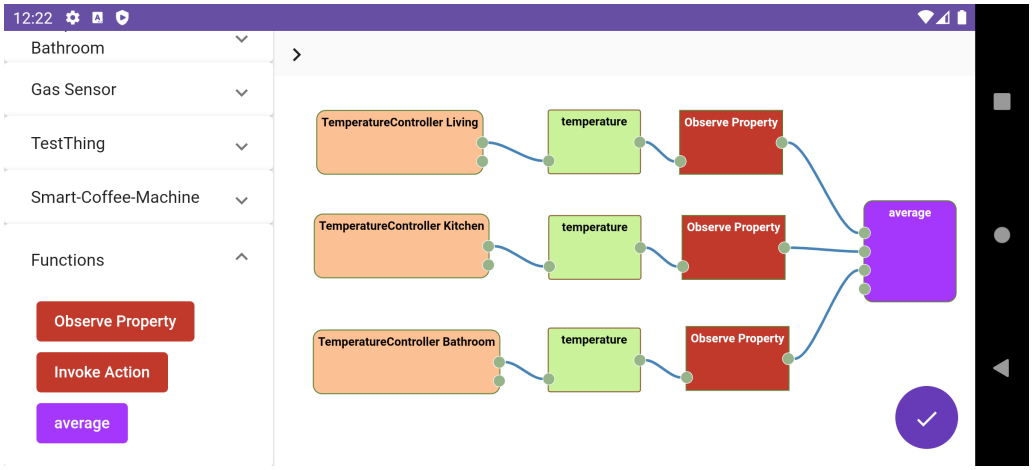


Figura 4.12: Modulo Frontend all'interno di una WebView

creazione di editor node-based. L'obiettivo di questo componente è quello di permettere la costruzione di un flusso di blocchi di diversi tipi e da esso generare del codice Java eseguibile a runtime direttamente su un dispositivo Android.

Al suo interno sono definiti una serie di nodi:

- ThingNode: rappresenta una Thing;
- ActionNode: rappresenta una Action Affordance, può richiedere dei valori in input;
- PropertyNode: rappresenta una Property Affordance;
- BasicFunctionNode: costituisce la base per gli InvokeActionNodes e gli ObservePropertyNodes;
- InvokeActionNode: se associato a un'ActionNode ne permette l'invocazione;
- ObservePropertyNode: se associato a un PropertyNode ne permette l'osservazione.
- ArithmeticFunctionNode: rappresenta una determinata funzione aritmetica (e.g. la media aritmetica).

La connessione tra nodi è sottoposta a regole precise - sono consentite le seguenti connessioni:

- ThingNode → ActionNode e ThingNode → PropertyNode: ad ogni Thing possono essere associate una o più proprietà e azioni.
- ActionNode → InvokeActionNode: solo le azioni possono essere invocate.
- PropertyNode → ObservePropertyNode: solo le proprietà possono essere osservate.
- ObservePropertyNode → ArithmeticFunctionNode: è possibile processare diverse proprietà numeriche tramite apposite funzioni aritmetiche (ad esempio la media aritmetica).

Una volta connessi i nodi desiderati, l'utente può salvare la Routine. Inizia quindi il processo di generazione del codice a partire dalla lista di nodi e dalla lista di connessioni, implementate dalla libreria *retejs*. Viene quindi creata una stringa a cui, ogni volta che un nodo viene analizzato, sono aggiunte delle righe di codice Java.

Prima di tutto si aggiungono alla stringa gli *import* necessari al corretto funzionamento del codice. Successivamente si verifica che sia disponibile all'interno dell'editor almeno un ThingNode e in caso affermativo si inizia a esplorare l'albero dei nodi tramite la funzione ricorsiva `inspectNextNode()`. In base al tipo di nodo esplorato vengono aggiunti alla stringa iniziale dei frammenti di codice differenti.

I valori ottenuti tramite gli ObservePropertyNode vengono aggiunti a un oggetto `Map` che poi viene richiesto a runtime dall'applicazione WoT Conductor che ne mostra il contenuto tramite la sua interfaccia. Quando viene esaminato un ObservePropertyNode, viene subito controllato anche il nodo successivo, e in caso sia un ArithmeticFunctionNode il valore della proprietà osservata (se numerico), viene aggiunto a un array che sarà processato al termine dell'esecuzione (e.g. viene fatta la media artimetica dei valori contenuti al suo interno).

Gli ActionNode al loro interno possono contenere una serie di informazioni accessorie, inserite dall'utente durante la creazione del grafo a blocchi. Queste informazioni corrispondono ai valori che vanno passati all'interno dei form della TD, come descritto dagli standard WoT, durante l'invocazione delle action (*uriVariables e inputs*). Vengono prima aggiunti uno ad uno ad un oggetto `Map` apposito che poi viene passato al metodo `invoke()` chiamato sulla Thing interessata.

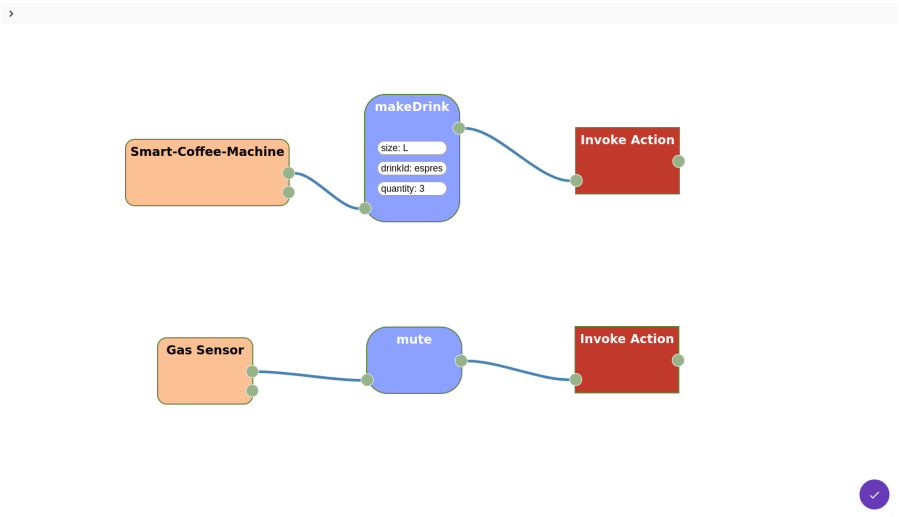


Figura 4.13: Routine con ActionNode semplice e ActionNode che richiede inputs

Una volta terminato il processo di creazione del codice, la Routine viene aggiunta al database tramite l'interazione con il modulo *api*.

Component *action-input-dialog*

Il component *action-input-dialog* consente all'utente, appena aggiunto all'editor un ActionNode, se necessario, di inserire i valori di input della relativa Action. Vengono infatti ottenute tutte le labels dei valori da inserire e vengono creati i relativi inputs, all'interno di un `MatDialog`.

Component *routine-name-dialog*

Il component *routine-name-dialog* permette l'inserimento del nome della Routine al momento della sua creazione. Una volta confermata la scelta del nome viene avviato il processo di generazione del codice all'interno del component *rete*.

Component *thing-expandable*

Il component *thing-expandable* viene abbinato ad una Thing. È un dropdown che contiene i diversi bottoni per l'aggiunta di ThingNodes, ActionNodes e PropertyNodes all'editor. Questo componente dialoga infatti con il component *rete* attraverso dei metodi specifici (e.g. `onAddPropertyNode()`).

Component *validation-alert*

Il component *validation-alert* consiste in un alert che segnala all'utente l'impossibilità di effettuare una connessione tra due specifici nodi. Si basa sul component NgBootstrap `NgbAlert`.

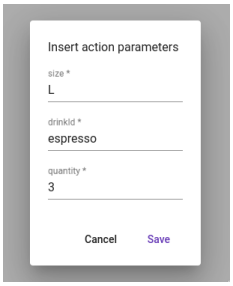
4.1.5 WoT Flow - modulo *api*

Il modulo *api* implementa un server NestJS per la gestione e l'esecuzione in Edge delle Routines. É associato alla porta 3001 del suo docker container.

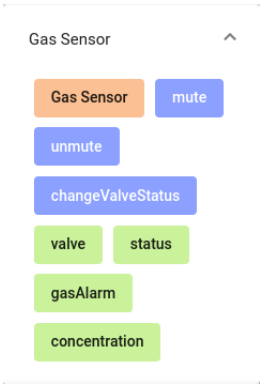
Modulo *routine*

All'interno del modulo *routine* é definito il `RoutineDto` carratterizzato dagli attributi `id, name, javaCode, tsCode, thingIds`, analoghi a quelli della classe Android `Routine`. Sono inoltre implementati anche il `RoutineController` e il `RoutineService` che definiscono i seguenti metodi per la gestione di richieste HTTP e i relativi funzionamenti:

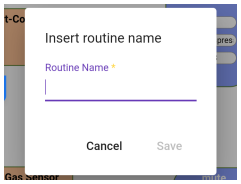
- `create()`: crea una Routine assegnandole un id univoco (UUID);
- `findOneBy()`: ritorna, se esiste, la Routine con ID fornito dall'utente;



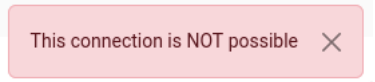
(a) Action Input Dialog



(b) Thing Expandable



(c) Routine Name Dialog



(d) Validation Alert

Figura 4.14: Componenti *frontend*

- `findAll()`: ritorna tutte le Routines salvate all'interno del database;
- `deleteOne()`: elimina dal database la Routine con ID corrispondente a quello passato dall'utente;
- `updateOne()`: sostituisce nel database la Routine con ID indicato dall'utente con il nuovo oggetto Routine passato dall'utente.

Modulo *execute*

Il modulo *execute* permette di eseguire sull'Edge una Routine e di ritornarne poi i risultati ottenuti come risposta. Il suo funzionamento si basa sul framework *node-wot* - un'implementazione degli standard WoT W3C per NodeJS. Include un *controller* che riceve la richiesta contenente un RoutineDto come body e un *service* che tramite un node-wot Servient esegue la Routine,

utilizzando la funzione `eval()` che permette di eseguire codice TypeScript o JavaScript direttamente da una stringa.

4.2 Tecnologie e librerie utilizzate

Android Native

Dovendo gestire numerosi dispositivi esterni è necessario che l'uso delle risorse sia ottimizzato al meglio. Sviluppare un'app nativa è la migliore scelta per garantire che i consumi siano ridotti al minimo, grazie all'integrazione diretta con hardware e sistema operativo. L'interazione con le funzionalità intrinseche del dispositivo è più immediata, inoltre, l'interfaccia utente segue le linee guida e gli standard del sistema operativo, offrendo un'esperienza utente coerente e familiare agli utenti.

Zion

Per una gestione efficiente delle Thing Descriptions e per l'implementazione del processo di Discovery è necessario l'utilizzo di una Thing Description Directory: una Thing che fornisce un servizio per gestire un insieme di TDs (Thing Descriptions) che descrivono altre Things.
La scelta di *Zion* è giustificata dal fatto che sia flessibile, facilmente integrabile, e che utilizzi tecnologie open-source.

Database Room

Il Database Room fornisce un'API di alto livello che semplifica notevolmente le operazioni di database rispetto alle API SQLite standard di Android. Riduce la quantità di codice necessario per gestire le operazioni di database. Intoltre, offre il supporto per le query reactive tramite l'integrazione con LiveData o RxJava. Ciò semplifica la gestione delle interfacce utente reattive e la sincronizzazione dei dati tra il database e l'interfaccia utente. È progettato per integrarsi facilmente con altri componenti delle Ar-

chitetture Android, come ViewModel e LiveData, facilitando la creazione di un'applicazione ben strutturata.

Material Design 3

Il design di Wot Conductor si basa su Material Design 3, sistema sviluppato da Google. Fornisce linee guida e componenti per la realizzazione di un'interfaccia grafica coerente e intuitiva, che offre all'utente un'esperienza fluida e uniforme.

Beanshell

Beanshell è un interprete Java open-source incorporabile anche in applicazioni Android. Permette di eseguire codice Java a runtime prendendo in input oggetti e contesti dal codice statico.

NSDManager

Per il processo di DNS Service Discovery viene utilizzata la libreria Android NSDManager. Fornisce un'API per la discovery e per la risoluzione dei servizi trovati in indirizzi IP e MAC. Permette inoltre la registrazione di un nuovo servizio sulla rete, con relativa pubblicizzazione. Questa feature non è però utilizzata nel progetto WoT Conductor.

GPT-4

GPT-4 (Generative Pre-trained Transformer - 4) è un modello di intelligenza artificiale generativa sviluppato da OpenAI. Costituisce il nucleo dell'assistente virtuale dell'applicazione ed è stato scelto per la sua larga diffusione e per la facilità di accesso alle sue API. È stato preferito a GPT-3.5 (la versione precedente) perchè offre risposte più accurate e precise. Al momento è una tecnologia a pagamento, il prezzo varia in base ai token (1K tokens $\sim$ 750 parole) in input e output: 0.01/1K tokens (input), 0.03/1K tokens (output).

oksse

Per la ricezione di Server Sent Events viene utilizzata la libreria *oksse* che è un'estensione della libreria *okhttp* (un client HTTP Java). Per rimanere in ascolto di eventi è sufficiente istanziare un client *okhttp* e abbinarlo a un *oksse* listener.

Material Expansion Panel

Material Expansion Panel è una libreria UI Android che implementa degli oggetti `Expandable` utilizzabili come dropdowns all'interno di una `RecyclerView`. In WoT Conductor sono usati nella schermata Discover per mostrare e nascondere azioni e proprietà relative ad ogni Thing. É una libreria open-source che permette un alto grado di customizzazione.

CodeView

Per una più semplice lettura del codice di ogni routine viene utilizzato il syntax-highlighter implementato dalla libreria *CodeView*, che riconosce diversi linguaggi di programmazione, come Java e TypeScript. Include anche un componente per il riconoscimento automatico del linguaggio utilizzato in una stringa di codice data.

Docker

Docker consente di eseguire applicazioni in ambienti isolati e facilmente distribuibili, semplificandone il *deployment*. Offre un sistema di *containerizzazione*, differente dalla classica *virtualizzazione*, in quanto con un docker-container la virtualizzazione viene effettuata a livello del sistema operativo e non dell'elaboratore nella sua totalità. In WoT Flow ogni modulo dell'applicazione viene containerizzato tramite *docker-compose*.

AngularJS

AngularJS è un framework per applicazioni web client-side, sviluppato da Google. In WoT Flow è utilizzato per l'implementazione del modulo *frontend*. È stato scelto perchè segue gli stessi design-patterns usati nello sviluppo Android Native, ovvero *MVC* (Model-View-Controller) e *MVVM* (Model-View-ViewModel).

NestJS

NestJS è un framework per applicazioni web server-side basate su Node-JS. È utilizzato per lo sviuluppo del modulo *api* di WoT Flow.

ReteJS

ReteJS è un framework JavaScript (scelto data l'inesistenza di librerie simili per Android/Java) che permette la creazione di editor node-based largamente personalizzabili, sia dal punto di vista UI che dal punto di vista funzionale. Tramite questa libreria è implementato il servizio di generazione di *routine-code* a partire da un diagramma di flusso (con un'interfaccia in stile Node-RED).

Angular Material

Per mantenere una coerenza tra lo stile dell'applicazione Android WoT Conductor (la cui interfaccia segue le linee guida descritte da Material Design 3) e la web-app WoT Flow, in quest'ultima, per l'implementazione della UI, sono usati i componenti della libreria Angular Material, che segue le specifiche del Material Design.

Capitolo 5

Validazione

Le funzionalità principali dell'applicazione sono quelle di generazione e di esecuzione di Routines. È importante quindi misurarne l'efficienza tramite l'analisi di metriche specifiche.

Per quanto riguarda la generazione del codice vengono confrontati GPT-3.5 e GPT-4, in modo da stabilire quale modello sia più adatto nel contesto del progetto sviluppato. Viene inoltre esaminato l'impatto che può avere il valore della *Temperature* durante il processo di generazione di codice.

Data la possibilità di eseguire il codice di una Routine in diversi ambienti, vengono creati due *use-cases* e vengono misurati i tempi di esecuzione delle relative Routines su device e in cloud, per osservare in quali casi sia opportuno preferire un ambiente rispetto ad un altro.

5.1 Metriche di generazione del codice

Sono diversi i modelli di Intelligenza Artificiale Generativa disponibili per la generazione di codice. I due modelli presi in considerazione durante lo sviluppo del progetto sono GPT-3.5 e GPT-4. Di seguito sono riportati dati e metriche raccolti durante la fase di testing del progetto.

5.1.1 GPT-3.5 vs GPT-4

Il confronto tra GPT-3.5 e GPT-4 viene fatto analizzando le seguenti metriche: corretto funzionamento della mashup generata, tempo impiegato per la generazione del codice, lunghezza del codice, tipologie di errore.

I dati sono ottenuti ripetendo 50 volte la generazione della stessa Routine (da eseguire sul device Android) a partire dal comando "*Voglio sapere la temperatura media del mio appartamento. Se è inferiore a 20 gradi preparami un thè.*", sia con GPT-3.5 che con GPT-4.

Sono coinvolte 4 Things: *TemperatureControllerLiving, TemperatureControllerKitchen, TemperatureControllerBathroom, SmartCoffeeMachine.*

Funzionamento della mashup generata

Ogni mashup application generata dalla AI viene eseguita e ne viene valutato il funzionamento. Le automazioni generate da GPT-3.5 che una volta eseguite portano a termine tutti i compiti previsti sono 4 su 50, mentre quelle generate da GPT-4 sono 37 su 50 [*Figura* 5.1]. È evidente che GPT-4 sia molto più preciso, anche su un campione di solo 50 prove.

Tempo impiegato per la generazione del codice

Viene misurato il tempo che passa dall'invio della richiesta all'arrivo della risposta dell'API OpenAI.

Il tempo medio impiegato da GPT-3.5 per generare il codice dell'automazione richiesta è di 17253 millisecondi, mentre per GPT-4 il valore medio corrisponde a 52964 millisecondi. Un risultato preciso e affidabile quindi corrisponde a un elevato tempo di generazione.

Lunghezza del codice

La lunghezza del codice corrisponde al numero di caratteri che lo compongono.

Il codice di una mashup application generata da GPT-3.5 è lungo in media 2273 caratteri, mentre quello generato da GPT-4 è composto mediamente di 2679 caratteri. In alcuni casi GPT-3.5 non ritorna codice, quindi la lunghezza considerata è di 0 caratteri, e non è considerata nel calcolo della media. In termini numerici la differenza non è significativa, ma in termini funzionali il codice generato da GPT-4 risulta essere molto più efficace.

Tipologie di errore

Nel caso in cui la mashup non funzioni in maniera adeguata viene raccolto il tipo di errore generato.

Nel codice prodotto da GPT-4 le tipologie di errori rilevate sono tre:

- uso di caratteri non riconosciuti dall'interprete *beanshell*;
- mancato display dei valori ottenuti dall'interazione con le Things;
- tentativo di casting da Float a String.

Durante l'esecuzione del codice generato da GPT-3.5 sono rilevate diverse tipologie di errore, oltre a quelle già descritte:

- mancata invocazione di Actions;
- mancata generazione di attributo della Routine (*Json field missing*);
- uso di oggetti non definiti in precedenza;
- mancanza di `imports`;
- errori di sintassi.

Il poco tempo impiegato da GPT-3.5 per la generazione del codice si traduce quindi in una bassa precisione che porta a errori considerabili banali come quelli di sintassi.

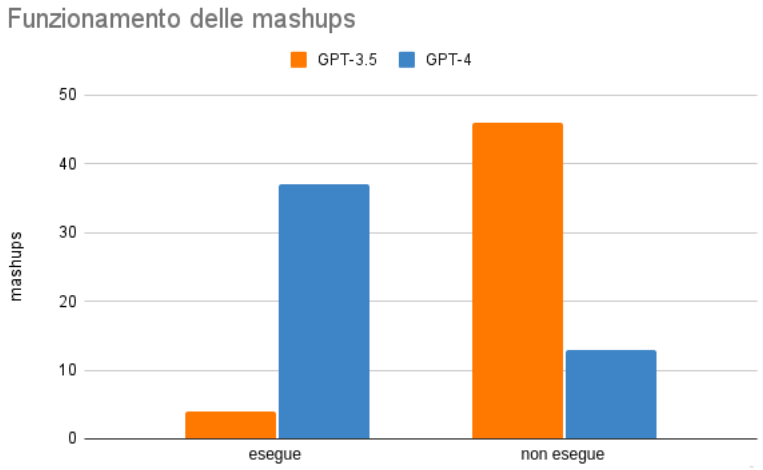
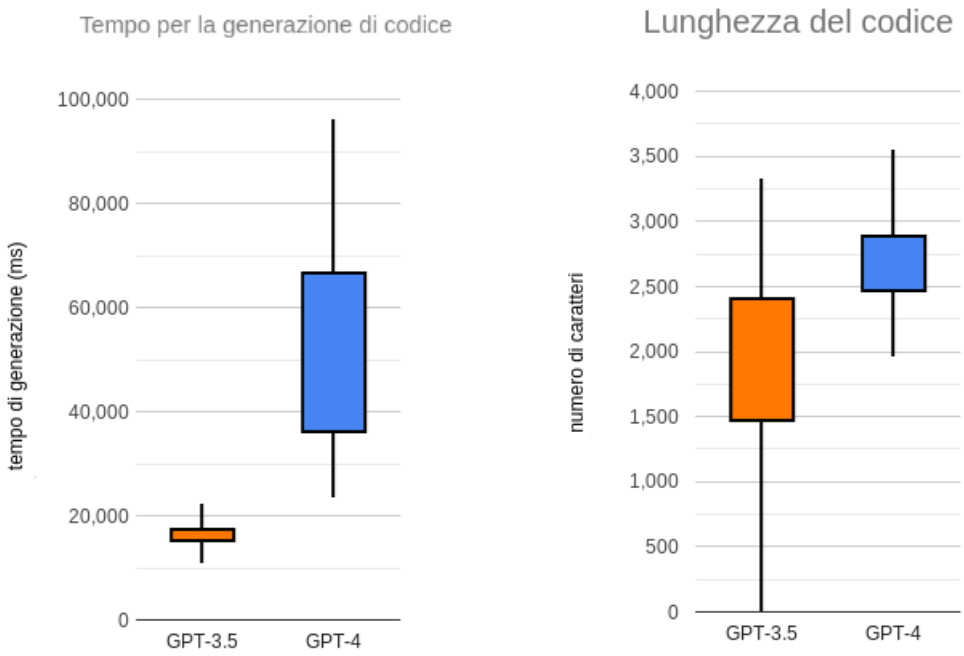


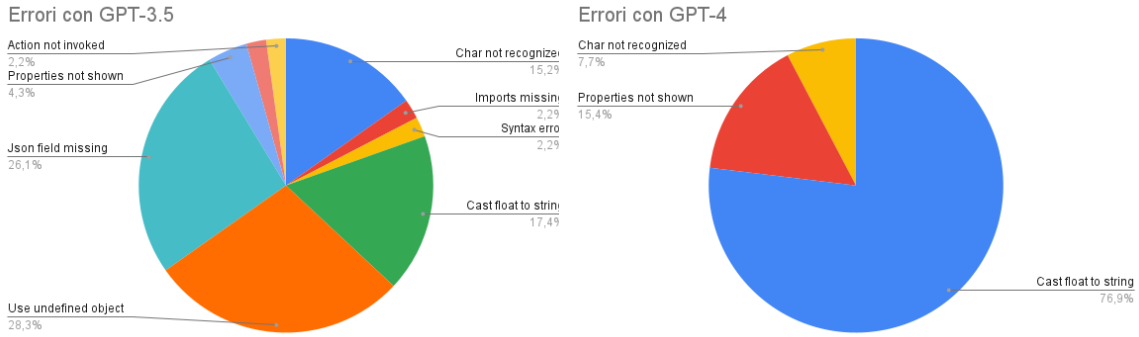
Figura 5.1: Funzionamento delle mashups



(a) Tempi di generazione

(b) Lunghezza del codice

Figura 5.2: Box Plots



(a) Errori rilevati con GPT-3.5 (b) Errori rilevati con GPT-4

Figura 5.3: Errori

5.1.2 Analisi sulla Temperature

La *Temperature* è un parametro utilizzato durante la generazione del testo e del codice che controlla la "conservatività" o "creatività" delle predizioni del modello. Una Temperature più bassa produce risposte più conservative e ripetitive, mentre una Temperature più alta genera risposte più creative e variegate [1].

Il codice per la Routine descritta al paragrafo 5.1.1, generato utilizzando GPT-4 con un valore di Temperature corrispondente a 0.2, risulta funzionante in 6 casi su 10, con un tempo di generazione medio di 54503 ms e una lunghezza media di 2693 caratteri.

Con un valore di Temperature uguale a 1.2, invece, il codice prodotto esegue in maniera corretta solo 3 volte su 10. La sua generazione richiede in media 61306 ms e ha una lunghezza media di 2795 caratteri.

Dai dati emerge quindi che per la generazione di codice sia necessario affidarsi a un modello più conservativo, che impiega meno tempo e riduce, seppur di poco, il numero di caratteri utilizzati, evitando di aggiungere funzionalitá superflue.

5.2 Metriche di esecuzione

L'esecuzione delle Routines può avvenire direttamente sul dispositivo Android oppure in remoto, tramite il servizio *api* fornito da WoT Flow. Per osservare le differenze tra i due metodi vengono eseguite due diverse Routines, in un ambiente composto da:

- Web Things esposte su un Raspberry Pi 3 nella rete locale domestica;
- servizi WoT Flow ospitati da una Google Cloud Virtual Machine;
- un dispositivo Android (connesso alla rete locale domestica).

5.2.1 Routine 1

La prima Routine ha un basso costo computazionale e interagisce con diverse Things. Corrisponde al seguente comando " *Voglio impostare la luminosità della lampadina a 0, spegnere il riscaldamento e chiudere la porta a chiave. Infine voglio controllare che la porta sia chiusa davvero.*"

L'automazione viene eseguita 1000 volte sul dispositivo Android e 1000 volte sul Google Cloud e ne vengono misurati i tempi di esecuzione [*Tabella* 5.1].

Metrica (ms)	**Cloud**	**Device**
Media	234.305	49.534
Massimo	1278	255
Minimo	194	36

Tabella 5.1: Tempi di esecuzione Routine 1

La differenza tra esecuzione diretta sul dispositivo e esecuzione sul cloud è notevole. Questo può essere imputato al fatto che il device Android sia connesso alla rete locale in cui sono esposte le Things e che quando si esegue una Routine in remoto si ha una latenza maggiore dovuta all'invio della richiesta al server e l'attesa della risposta.

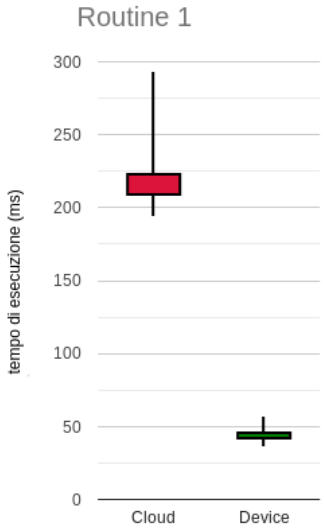


Figura 5.4: Tempi di esecuzione Routine 1

5.2.2 Routine 2

La seconda Routine è progettata in modo da risultare computazionalmente onerosa. Essa consiste nell'azzeramento della luminosità di una lampadina, per avere comunque un'interazione con una Thing, e nel calcolo del prodotto di due matrici 100×100, operazione con un costo computazionale elevato. Viene eseguita 1000 volte sul device Android e 1000 volte sul cloud. I tempi di esecuzione sono mostrati nella *Tabella 5.2* e nella *Figura 5.5*.

Metrica (ms)	**Cloud**	**Device**
Media	106.113	7045.215
Massimo	355	10253
Minimo	82	6278

Tabella 5.2: Tempi di esecuzione Routine 2

Al contrario di ciò che accade nell'automazione illustrata precedentemente, l'esecuzione in cloud di questa Routine è molto più veloce rispetto a quella

sul device Android. Questo può essere giustificato dal fatto che la potenza di calcolo disponibile sul cloud sia maggiore rispetto a quella del dispositivo Android, che oltre a eseguire la Routine deve anche occuparsi di altre operazioni.

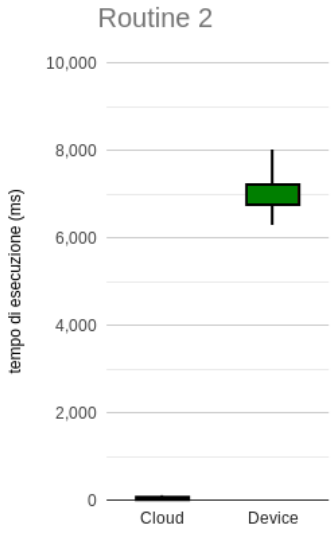


Figura 5.5: Tempi di esecuzione Routine 2

5.2.3 Risultati

Dai dati raccolti emerge come, in base alle esigenze, sia necessario affidarsi a un diverso tipo di esecuzione. Per automazioni relativamente semplici e con basso costo computazionale è conveniente optare per l'esecuzione diretta sul dispositivo, mentre per Routines computazionalmente dispendiose l'esecuzione sul cloud si dimostra una soluzione efficiente.

Capitolo 6

Conclusioni

In questo elaborato è stata illustrata l'implementazione di un'applicazione *Android Native* e di servizi accessori per l'orchestrazione di diversi dispositivi IoT. Sono state esaminate le tecnologie utilizzate durante lo sviluppo del progetto: partendo dall'analisi del paradigma *Internet of Things* si è arrivati a vedere come sia stato necessario introdurre gli standard del *Web of Things* per attenuare i problemi di interoperabilità che hanno da sempre caratterizzato l'IoT, per poi concentrarsi sui modelli più diffusi nel mondo dell'*Intelligenza Artificiale Generativa*. Sono state infine misurate l'efficienza e la precisione del software implementato, dimostrando come quest'ultimo sia utilizzabile in diversi contesti e per diversi scopi.

6.1 Sviluppi futuri

Essendo l'applicazione molto articolata sono diverse le migliorie che possono essere sviluppate. In primo luogo, si potrebbe sfruttare la possibilità di esporre Things data dall'implementazione dell'*AndroidWotServient* per interagire con tutti i sensori incorporati in ogni device Android.

Un aspetto da approfondire nel futuro è indubbiamente quello della sicurezza. Il codice di ogni Routine infatti, al momento viene eseguito direttamente a runtime senza controlli specifici. Potrebbe quindi essere iniet-

tato all'interno dell'applicazione codice malevolo. Risulta perciò necessaria un'analisi più accurata di questo tema.

Visto che l'*AndroidWotServient* permette di interagire anche con gli eventi generati da una Thing, oltre che con le azioni e le proprietà, si potrebbe aggiungere al modulo *frontend* di WoT Flow un nodo dedicato e tutto ciò che ne consegue. Oltre a ciò, l'intrefaccia dell'editor a blocchi potrebbe essere migliorata e resa più gradevole sia da un punto di vista funzionale che visivo, dato che le opzioni di personalizzazione fornite dal framework *ReteJS* sono svariate.

Il tempo di generazione del codice da parte di GPT-4 (oltre che al costo), sono aspetti che rendono l'applicazione meno appetibile e che potrebbero essere migliorati tramite l'integrazione di un modello di generazione di codice progettato direttamente per questo utilizzo.

Il progetto sviluppato è quindi una base sulla quale si possono costruire molte altre funzionalità, in modo da rendere l'applicazione ancora più versatile.

Bibliografia

[1] OpenAI API. OpenAI Platform. `https://platform.openai.com/docs/api-reference/chat`.

[2] P Abdul Hafeez, Gurpreet Singh, Jasrpreet Singh, Chander Prabha, and Amit Verma. Iot in agriculture and healthcare: Applications and challenges. In *2022 3rd International Conference on Smart Electronics and Communication (ICOSEC)*, pages 446–450, 2022.

[3] Saleh Hussein Al-Awami, Mohammed Mahfud Al-Aty, and Mousa Faraj Al-Najar. Comparison of iot architectures based on the seven essential characteristics. *IEEE Access*, pages 305–310, 2023.

[4] Martina Antonic. Iot technologies offer new potentials for people with disabilities. In *2021 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–6, 2021.

[5] Kevin Ashton. That "internet of things" thing — RFID journal, June 2009.

[6] Ekaterina Balandina, Sergey Balandin, Yevgeni Koucheryavy, and Dmitry Mouromtsev. Iot use cases in healthcare and tourism. In *2015 IEEE 17th Conference on Business Informatics*, volume 2, pages 37–44, 2015.

[7] Vinicius Barros, Sergio Junior, Sarita Bruschi, Francisco Monaco, and Julio Estrella. An iot multi-protocol strategy for the interoperability of

distinct communication protocols applied to web of things. pages 81–88, 10 2019.

[8] Staphord Bengesi and et al. Advancements in generative ai: A comprehensive review of GANs, GPT, autoencoders, diffusion model, and transformers. *ArXiv.org*, 2023.

[9] Andre Mendes Cavalcante, Pedro Henrique Gomes, Maria Valeria Marquezini, Iracema Bonomini, and Luciano Leonel Mendes. Applicability of iot technologies for 5g use cases in brazil. In *2019 IEEE 2nd 5G World Forum (5GWF)*, pages 53–57, 2019.

[10] Asli Celikyilmaz and et al. Evaluation of text generation: A survey. *ArXiv*, 2021.

[11] Enrique Dehaerne, Bappaditya Dey, Sandip Halder, Stefan De Gendt, and Wannes Meert. Code generation using machine learning: A systematic review. *IEEE Access*, 10:82434–82455, 2022.

[12] Asmae El Jaouhari, Meriem Azari, Jabir Arif, Imane Ibn El Farouk, Fouad Jawab, and Imane Moufad. Iot for the future of sustainable supply chain management in industry 4.0: A systematic literature review. In *2022 14th International Colloquium of Logistics and Supply Chain Management (LOGISTIQUA)*, pages 1–6, 2022.

[13] Mohammed Elkahlout, Mohammed M. Abu-Saqer, Ahmed Fadl Aldaour, Ahmed Issa, and Mojca Debeljak. Iot-based healthcare and monitoring systems for the elderly: A literature survey study. In *2020 International Conference on Assistive and Rehabilitation Technologies (iCareTech)*, pages 92–96, 2020.

[14] Konstantina-Maria Giannakopoulou, Ioanna Roussaki, and Konstantinos Demestichas. Internet of things technologies and machine learning methods for parkinson; disease diagnosis, monitoring and management: A systematic review. *Sensors*, 22(5), 2022.

[15] Google. Overview of GAN structure — generative adversarial networks. `https://developers.google.com/machine-learning/gan/gan_structure`, 2019.

[16] Google Developers. The discriminator — generative adversarial networks. `https://developers.google.com/machine-learning/gan/discriminator`, 2019.

[17] Roberto Gozalo-Brizuela and Eduardo C. Garrido-Merchan. ChatGPT Is Not All You Need. A State of the Art Review of Large Generative AI Models. *ArXiv*, 2023.

[18] Dominique D. Guinard and Vlad M. Trifa. *Building the Web of Things*, volume 3. Manning Publications, Shelter Island, NY, USA, 2016.

[19] Baskhad Idrisov and Tim Schlippe. Program code generation with generative ais. *Algorithms*, 17(2), 2024.

[20] Satyabrata Jena. Architecture of Internet of Things (IoT), June 2020.

[21] M. Jovanovic and M. Campbell. Generative artificial intelligence: Trends and prospects. *Computer*, 55(10):107–112, oct 2022.

[22] Abdulkadir Karaagac, Niels Verbeeck, and Jeroen Hoebeke. The integration of lwm2m and opc ua: An interoperability approach for industrial iot. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pages 313–318, 2019.

[23] Adrian Kast, Ege Korkan, Sebastian KÃ¤bisch, and Sebastian Steinhorst. Web of things system description for representation of mashups. In *2020 International Conference on Omni-layer Intelligent Systems (COINS)*, pages 1–8, 2020.

[24] Euijong Lee, Young-Duk Seo, Se-Ra Oh, and Young-Gab Kim. A survey on standards for interoperability and security in the internet of things. *IEEE Communications Surveys & Tutorials*, 23(2):1020–1047, 2021.

[25] Prathyusha M R and Biswajit Bhowmik. Development of iot-based smart home application with energy management. In *2023 International Conference on Sustainable Communication Networks and Application (ICSCNA)*, pages 367–373, 2023.

[26] Lei Mao. Minmax game for training generative adversarial networks. `https://leimao.github.io/blog/Generative-Adversarial-Networks-Minmax-Game/`, October 2020.

[27] Media, OpenSystems. Introduction to web of things (wot): Here's everything you need to know, January 2022.

[28] Mahda Noura, Mohammed Atiquzzaman, and Martin Gaedke. Interoperability in internet of things: Taxonomies and open challenges. *Mobile Networks and Applications*, 24(3):796–809, June 2019. DOI: 10.1007/s11036-018-1089-9.

[29] Yangchen Palmo, Shigeaki Tanimoto, Hiroyuki Sato, and Atsushi Kanai. Iot reliability improvement method for secure supply chain management. In *2021 IEEE 10th Global Conference on Consumer Electronics (GCCE)*, pages 364–365, 2021.

[30] Brien Posey. What are IoT devices?, August 2023.

[31] Fariza Sabrina. Blockchain and structural relationship based access control for iot: A smart city use case. In *2019 IEEE 44th Conference on Local Computer Networks (LCN)*, pages 137–140, 2019.

[32] Giulio Salvadori and et al. La crescita dell'internet of things: Mercato, applicazioni e nuovi servizi, April 2022.

[33] Salvadori, Giulio. Mercato IoT italia 2023: Giro d'affari di 8,3 miliardi di euro e 124 milioni di oggetti connessi, April 2023.

[34] Luca Sciullo and et al. WoT Store: Managing resources and applications on the web of things. *Internet of Things*, 9:100164, March 2020.

[35] Luca Sciullo and et al. A survey on the web of things. *IEEE Access*, 10:47570–47596, January 2022.

[36] Sneha and et al. Narrow band-IoT and long-range technology of IoT smart communication: Designs and challenges. *Computers & Industrial Engineering*, 172:108572, October 2022.

[37] Burkhard Stiller and et al. An overview of network communication technologies for IoT, Year.

[38] Joni Turunen and Aleksanteri Fagerholm. Technology report: From ideas to applications, 12 2023.

[39] Lionel Sujay Vailshery. Iot connected devices worldwide 2019-2030, July 2023.

[40] Ashish Vaswani and et al. Attention is all you need. *ArXiv.org*, 2017.

[41] History - web of things (WoT). `https://www.w3.org/WoT/about/history/`.

[42] Web of Things (WoT) Binding Templates. `https://www.w3.org/TR/wot-binding-templates/`.

[43] Web of Things (WoT) Architecture 1.1. `https://www.w3.org/TR/wot-architecture11/`, Dec 2023.

[44] Web of Things (WoT) Scripting API. `https://www.w3.org/TR/wot-scripting-api/`, Oct 2023.

Ringraziamenti

Ringrazio il mio relatore Dott. Luca Sciullo per la sua disponibilità e per avermi permesso di realizzare questo progetto.
Ringrazio i miei genitori per avermi sempre sostenuto nelle mie scelte e in questo percorso.
Ringrazio i miei nonni e i miei zii per la fiducia che continuamente ripongono in me.
Ringrazio i miei amici di sempre per la loro costante presenza e vicinanza.
Ringrazio i nuovi amici e i Sunday House per avermi aiutato a vedere le cose anche da un'altra prospettiva.