Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Design e sviluppo di un W3C servient Android tramite l'utilizzo della libreria kotlin-wot

Relatore:
Dott.
LUCA SCIULLO

Presentata da:
FILIPPO LUPPI

Sessione I
Anno Accademico 2024/2025

Abstract

La rapida crescita dell'Internet of Things ha generato una frammentazione significativa nell'ecosistema, con protocolli eterogenei e standard proprietari che limitano l'interoperabilità tra dispositivi. Per rispondere a questa problematica, il World Wide Web Consortium (W3C) ha sviluppato le specifiche del Web of Things (WoT), che utilizzano i principi del web per unificare tecnologie esistenti senza crearne di nuove. Tuttavia, tra le implementazioni ufficialmente riconosciute dal W3C, nessuna è dedicata specificatamente al mondo Android mobile.
Questa tesi presenta Android WoT Servient, un Servient WoT completo per dispostivi Android basato sul framework kotlin-wot. Il progetto usa lo smartphone per hostare un server WoT, esponendo i sensori del dispostivo come Interaction Affordances standardizzate attraverso Thing Description. Il sistema supporta multipli protocolli di comunicazione (HTTP, MQTT, WebSocket).
La validazione sperimentale ha quantificato l'overhead delle specifiche WoT attraverso test comparativi. L'analisi costi-benefici dimostra che i vantaggi in termini di standardizzazione e interoperabilità giustificano l'investimento computazionale.

Indice

Capitolo 1

Introduzione

L'*Internet of Things* (IoT) ha trasformato radicalmente il panorama tecnologico contemporaneo, connettendo miliardi di dispositivi e generando un ecosistema digitale senza precedenti[1]. Tuttavia, la crescita esponenziale di questo settore ha portato a una frammentazione significativa, con standard di comunicazione e protocolli che spesso non sono interoperabili tra loro. Questa proliferazione di soluzioni proprietarie comporta un aumento dei costi di sviluppo e implementazione, oltre a creare notevoli difficoltà nell'integrazione di dispositivi provenienti da fornitori diversi[2].
Per rispondere a queste problematiche critiche, Il *World Wide Web Consortium* (W3C)[3] ha sviluppato le specifiche del *Web of Things*, un framework standardizzato che mira a unificare l'ecosistema IoT attraverso l'adozione di tecnologie web consolidate. L'approccio WoT estende il Web esistente per includere oggetti del mondo reale e dispositivi embedded, utilizzando HTTP come livello applicativo e rendendo le funzionalità dei dispositivi accessibili tramite API RESTful standardizzate.
I dispositivi Android rappresentano un'opportunità unica per l'espansione dell'ecosistema WoT. Con oltre il 70% del mercato globale degli smartphone [4], Android offre una base installata enorme che può essere sfruttata per creare una rete distribuita di Things interconnessi. I moderni smartphone Android integrano una ricchezza di sensori senza precedenti, uniti a fotocamere ad alta risoluzione e microfoni di qualità professionale.
Questa varietà di capacità sensoriali, combinata con processori potenti e connettività disponibile ovunque (WiFi, 4G, 5G, Bluetooth), trasforma ogni dispositivo in una potenziale stazione di monitoraggio ambientale completa, capace di raccogliere dati multidimensionali sull'ambiente circostante e di elaborarli localmente attraverso tecniche di *edge computing*.
Questo lavoro di tesi si propone di sviluppare un Servient WoT completo per dispositivi Android, trasformando gli smartphone in nodi attivi dell'Internet of Things conformi alle specifiche W3C. Il progetto mira a dimostrare la fattibilità di implementare un server WoT mobile che esponga i sensori del dispositivo come *Interaction Affordances* standardizzate, rendendo accessibili le funzionalità dello smartphone tramite protocolli web

standard.
I risultati dimostrano che, nonostante l'overhead introdotto dalle specifiche WoT, i benefici in termini di standardizzazione e interoperabilità giustificano l'investimento computazionale.
Il documento è organizzato nel seguente modo. Nel capitolo 2 viene presentato lo stato dell'arte, analizzando l'evoluzione dell'Internet of Things fino al Web of Things, descrivendo l'architettura e i building blocks delle specifiche W3C, e introducendo i framework di implementazione esistenti. Nel capitolo 3 viene definita l'architettura del progetto Android WoT Servient, specificando obiettivi, requisiti tecnici e funzionali, e descrivendo i principali componenti del sistema. Nel capitolo 4 viene documentata l'implementazione dettagliata, analizzando la struttura del codice, le tecnologie utilizzate e le scelte progettuali adottate per garantire performance e affidabilità. Infine, nel capitolo 5, viene presentata la validazione sperimentale attraverso test di performance comparativi, quantificando l'overhead delle specifiche WoT e fornendo raccomandazioni per l'uso ottimale del sistema.

Capitolo 2

Stato dell'Arte

2.1 Internet of Things

Con *Internet of Things* si descrive la rete di oggetti fisici, anche chiamate *"things"*, che hanno sensori, software e altre tecnologie integrate allo scopo di connettere e scambiare dati con altri dispositivi e sistemi su Internet.
Il termine viene coniato nel 1999 da Kevin Ashton, co-fondatore dell'Auto-ID Center del MIT, durante una presentazione in cui immaginava "un mondo dove gli oggetti potessero comunicare autonomamente attraverso la tecnologia RFID". [5] L'IoT si basa sull'idea di collegare al mondo digitale gli oggetti della nostra esperienza quotidiana: non solo dispositivi tecnologici come computer o smartphone, ma anche di oggetti comuni come elettrodomestici, auto, semafori e persino abbigliamento. Questi oggetti, una volta dotati di sensori e connettività, diventano "smart", in grado di raccogliere dati dall'ambiente circostante, elaborarli e comunicare autonomamente tra loro e con sistemi centrali.[6]
Partendo dall'idea di ogni oggetto che può potenzialmente diventare "smart", si è arrivati all'applicazione di IoT in molti ambiti: tra i principali abbiamo: Smart Home, Smart Car, Smart City, Smart Agriculture e Smart Factory.

2.1.1 Architettura

La moltitudine di dispositivi diversi che comprende la famiglia dell'IoT ha reso necessario un'architettura standardizzata. Il modello base è quello a 3 layers [7]:

- **Perception Layer**: si interfaccia col mondo esterno tramite sensori e attuatori per raccogliere dati grezzi.
- **Network/Transport Layer**: si occupa di trasmettere i dati all'*Application Layer* utilizzando diverse tecnologie (RFID, Wi-Fi, Bluetooth...).

- **Application Layer**: fornisce un'interfaccia all'utente visualizzare i dati e interagire col sistema.

Oltre al modello a 3 layers, troviamo comune anche un modello a 5 layers, con l'aggiunta di:

- **Middleware Layer/Processing Layer**: salva, analizza e pre-processa i dati in arrivo dal *Transport Layer*. Questo include attività come aggregazione dati, traduzione dei protocolli e rafforzamento della sicurezza per preparare i dati per l'*Application Layer*.
- **Business Layer**: User interface, dashboard e tool di visualizzazione dati per le aziende.

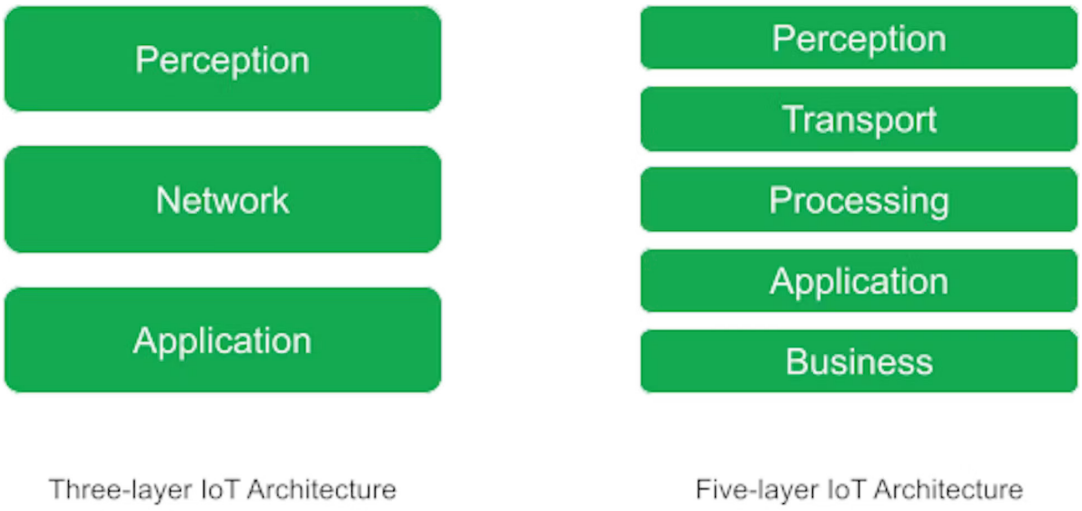

Figura 2.1: Modello a 3 o 5 layers[7]

2.1.2 Applicazioni

Negli anni il numero dei dispositivi connessi è cresciuto esponenzialmente, e unito allo sviluppo tecnologico, ha portato l'IoT in vari ambiti. Vediamo alcuni dei principali[8]:

- *Smart Home*: attraverso interruttori, sensori e dispositivi intelligenti che comunicano su protocolli come Zigbee, i sistemi di automazione domestica possono essere usati per monitorare e azionare, anche da remoto, l'illuminazione, la climatizzazione, i sistemi di sicurezza, gli elettrodomestici e altro ancora.
- *Smart City*: secondo lo Smart City Index[9] la città intelligente è *un ambiente urbano che applica la tecnologia per esaltare i vantaggi e attenuare gli inconvenienti*

dell'urbanizzazione. Ad esempio installando sensori nelle calditoie è possibile rilevare i livelli dell'acqua e automatizzare le azioni mirate a prevenire le inondazioni non appena i valori superano una certa soglia.

- *Smart Car*: i sistemi avanzati alla guida (*ADAS*) che si avvalgono della tecnologia IoT aiutano gli automobilisti a evitare incidenti, pianificare gli itinerari, parcheggiare in autonomia.
- *Smart Agriculture*: monitorare le condizioni del terreno, i modelli metereologici e la crescita delle colture. Ad esempio usare i sensori per misurare il contenuto di umidità del terreno, garantendo che le colture siano irrigate al momento giusto.
- *Smart Factory*: monitoraggio della linea di produzione per consentire la manutenzione proattiva delle apparecchiature quando i sensori rilevano un guasto imminente.

Questi sono solo gli esempi di applicazioni principali, ma ormai l'IoT si sta inserendo in ogni settore.

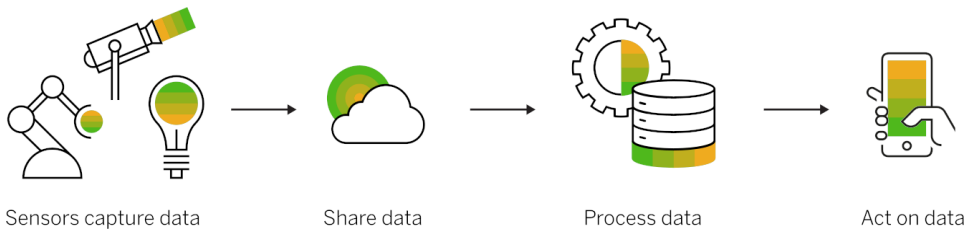

Figura 2.2: Le quattro fasi principali[8]

2.1.3 Protocolli

Sono stati introdotti diversi protocolli di comunicazione nel corso del tempo, ognuno adatto a dispositivi diversi: gli oggetti IoT più piccoli non hanno la stessa batteria di uno smartphone e necessitano quindi di ottimizzare il più possibile l'uso di energia.
Vediamo i più importanti:

- **Hyper Text Transfer Protocol (HTTP)**: protocollo ASCII puramente testuale basato sul modello client/server di richiesta/risposta tramite gli URI. Spesso ha un header troppo pesante per i device più "leggeri".
- **Constrained Application Protocol (CoAP)**: protocollo aperto e leggero progettato da IETF (Internet Engineering Task Force)[10], simile al linguaggio HTTP ma riprogettato per soddisfare i requisiti dei dispositivi con risorse CPU e RAM limitate. Utilizza pacchetti di dati molto piccoli e si basa su un modello client/server.

- **Message Queuing Telemetry Transport (MQTT)**: standard open di OASIS (*Organization for the Advancement of Structured Information Standards* utilizzato per gli scambi di dati leggeri. Si basa sul meccanismo publish/subscribe ed è adatto negli ambienti con poca banda a disposizione.
- **WebSocket**: progettato anche questo da IETF, per fornire comunicazione *full-duplex* tra web browser e server per sostituire HTTP riuscendo comunque a usare la sua struttura già esistente. Recentemente usato per le applicazioni IoT che necessitano di comunicazione real-time.

2.2 Web of Things

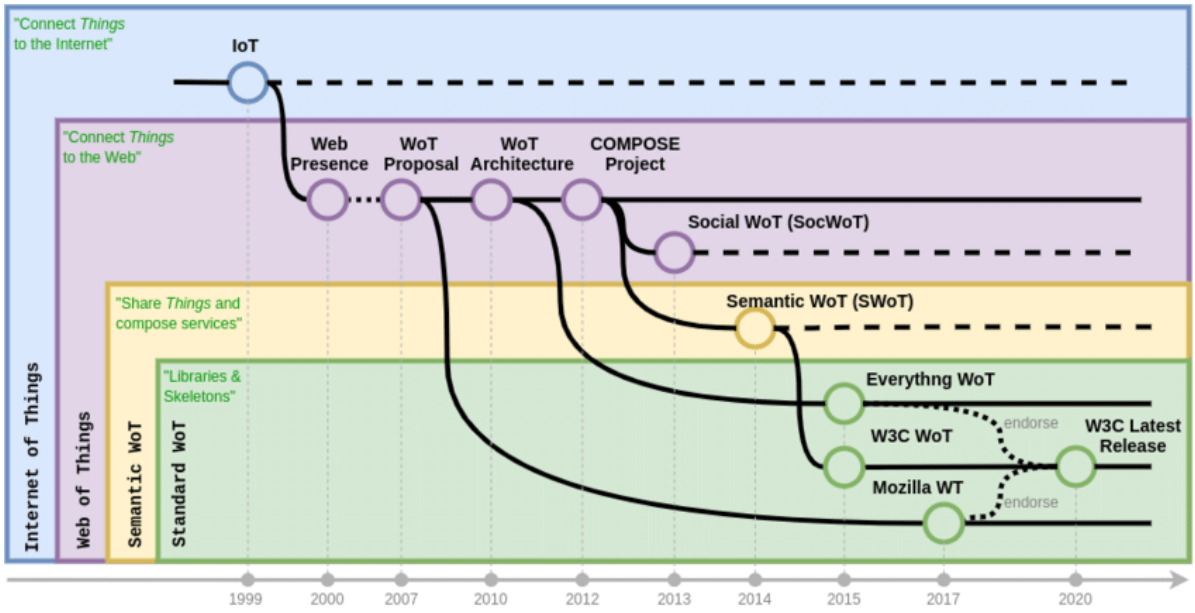

Figura 2.3: Evoluzione del Web of Things nel tempo[11]

La crescita esponenziale di dispositivi connessi ha portato a una frammentazione del mercato, con standard di comunicazione e protocolli diversi che spesso non sono interoperabili tra loro: questo comporta un aumento dei costi di sviluppo e implementazione, oltre a creare difficoltà nell'integrazione di dispositivi di fornitori diversi. Per rispondere a queste problematiche è emersa la necessità di sviluppare un linguaggio comune e standard unificati, dando vita al concetto di **Web of Things**.
L'idea nasce nel 2008 quando due ricercatori, Dominique Guinard e Vlad Trifa[12], pubblicano il primo *manifesto WoT*, in cui sostengono l'uso degli standard Web REST per costruire l'Application Layer. L'idea era quella di estendere il Web esistente includendo oggetti del mondo reale e dispositivi embedded, usando *HTTP* come livello applicativo e rendendo le funzionalità dei dispositivi disponibili tramite le **API RESTful**, che consentono tramite URI (Uniform Resource Identifier) di identificare singolarmente una risorsa e poi fornisce il supporto per un insieme di interazioni:

- **PUT**: aggiorna una risorsa esistente
- **GET**: recupera una risorsa
- **POST**: crea una nuova risorsa

- **DELETE**: elimina una risorsa

Guinard e Trifa hanno identificato due strade principali per raggiungere questo: *Direct Integration* e *Indirect Integration*[11]: nel primo approccio i dispositivi incorporano direttamente un server web diventando parte del Web, mentre nell'altro si ha un intermediario nello *Smart Gateway*, incaricato di tradurre le richieste e risposte REST attraverso protocolli in istruzioni specifiche per i dispositivi.

2.2.1 L'avvento del W3C nel WoT

Nel 2014, il *World Wide Web Consortium* (W3C) manifesta interesse nel WoT e crea il **WoT Interest Group**, che porta al primo ufficiale modello per Thing WoT. Da quel momento la popolarità del progetto cresce e il modello viene aggiornato in modo costante fino alle ultime specifiche che rappresentano lo standard *de-facto* per ogni componente dell'ecosistema Web of Things.
Il concetto centrale è la **Thing**. Essa è *un'astrazione di un'entità fisica o virtuale i cui metadata e interfacce sono descritte da* ***WoT Thing Description***, uno dei 6 *building blocks* necessari all'implementazione dell'architettura del W3C WoT.

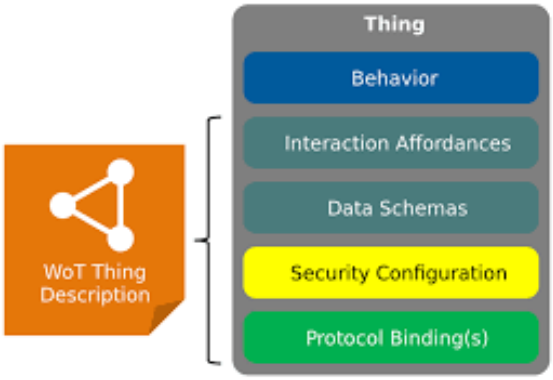

Figura 2.4: Thing Description[13]

Architettura

Il *Web of Things* si basa su due approcci architetturali principali: le architetture **stratificate** *(layered architectures)* e le architetture **Thing-centriche**.[11]
L'**architettura stratificata** più prominente è quella proposta da Guinard e Trifa[14], che decompone le funzionalità di un sistema WoT in layer separati e indipendenti, costruiti sorpa le tecnologie di comunicazione M2M di base e i protocolli IoT. Ogni layer può essere implementato all'interno di un singolo dispositivo intelligente, nei gateway o attraverso più nodi di rete. Vediamo ogni layer più nel dettaglio:

- **Access Layer**: responsabile di trasformare ogni Thing in una *Web* Thing, fornendo a ciascun dispositivo API rest per supportare le interazioni con il mondo esterno. Questo layer permette alle Thing di diventare interagibili tramite richieste HTTP come qualsiasi altra risorsa nel Web. Per superare la limitazione della natura richiesta/risposta di HTTP negli scenari IoT basati sugli eventi, si suggerisce l'uso di web sockets.

- **Find Layer**: garantisce che le Web Thing (WT) possano essere automaticamente scoperte e utilizzate, minimizzando la necessità di configurazione manuale. Si basa sul web semantico, permettendo di descrivere le Thing e i loro servizi usando standard semantici e interfacce uniformi. In questo modo le Thing diventano rintracciabili tramite i motori di ricerca e possono essere utilizzate automaticamente da altre applicazioni WoT.

- **Share Layer**: definisce come i dati prodotti dalle WT possano essere condivisi con utenti/applicazioni esterni in modo sicuro ed efficiente. Utilizza tutti i meccanismi di sicurezza Web (come TLS, OAuth) e può sfruttare piattaforme *SocWoT* per massimizzare l'estensione della disseminazione dei dati.

- **Compose Layer**: include soluzioni per implementare applicazioni WoT Mashup collegando insieme dati e servizi da diverse WT. Questo layer è responsabile di facilitare la creazione di applicazioni composite, avvicinando non solo gli sviluppatori ma anche gli utenti finali, integrando servizi e dati delle Thing in applicazioni web più complesse in modo semplice.

Diversamente dalle architetture stratificate, gli approcci **Thing-centrici** considerano le WT come un'unità coesa di dati e funzionalità. Lo standard **W3C WoT**[13] rappresenta la soluzione di riferimento per implementare sistemi WoT. Ogni WT è caratterizzata da **cinque moduli principali**:

- **Behavior**: rappresenta la logica di business della WT, dove sono implementati sia il comportamento autonomo della Thing che i gestori per le WT Affordances.

- **Interaction Affordances**: rappresentano il modello di interazione delle WT, specificando come i client WoT possono interagire con essa attraverso operazioni astratte, seguendo il paradigma *Properties, Actions and Events*.

- **Data Schemas**: rappresentano il modello di informazione da utilizzare nell'interazione tra la WT e i suoi consumatori.

- **Security Configuration**: contiene i meccanismi di sicurezza forniti dalla WT per controllare l'accesso alle sue Interaction Affordances.

- **Protocol Bindings**: mappano le Interaction Affordance ai messaggi di un protocollo di rete specifico.

I primi quattro moduli costituiscono la **Thing Description** (TD), una collezione di metadati che rappresenta le capacità della WT in modo uniforme, permettendo a qualsiasi client WoT di istanziare un collegamento di comunicazione con la WT dopo aver analizzato la TD.

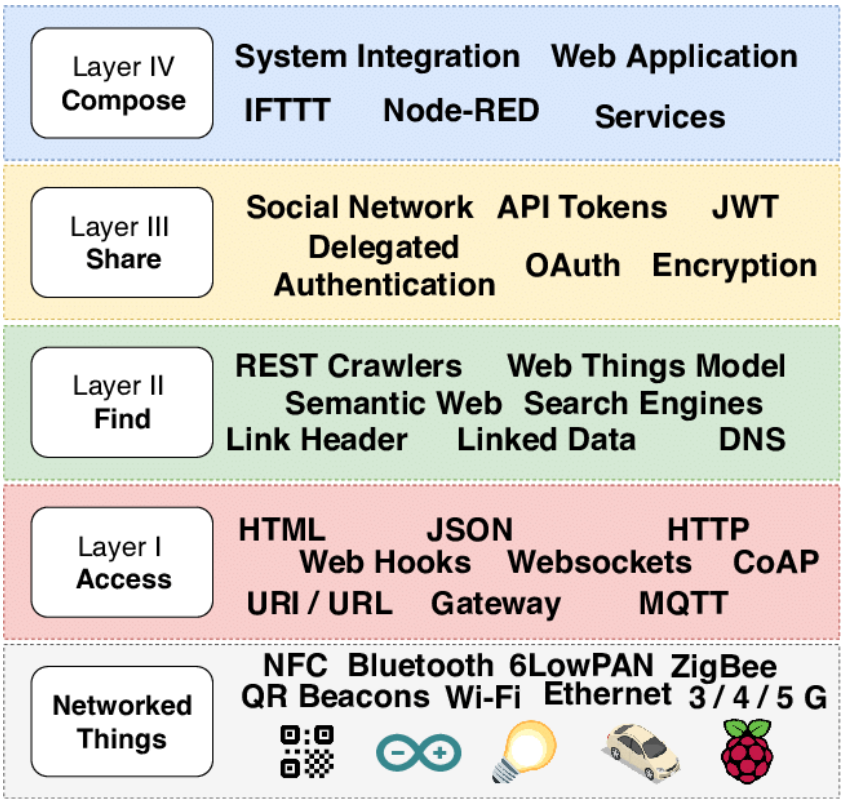

Figura 2.5: Architettura Web of Things[11]

Interaction Affordance

Le Interaction Affordances sono una componente fondamentale dello standard W3C Web of Things e rappresentano *la parte operativa di una TD*. Il loro scopo è definire come i client del WoT possono interagire con una *Web Thing* attraverso operazioni astratte, senza fare riferimento a un protocllo di rete specifico. Ne esistono di 3 tipi:

- **Proprietà**: una proprietà espone lo stato di una Thing. Questo stato può essere leggibile e, opzionalmente, può essere aggiornabile. Esempi di proprietà sono valori di un sensore (read-only), attuatori con stato (read-write), stato della Thing (read-only o read-write).

- **Azioni**: un'azione permette di eseguire una funzione sulla Thing. Un'azione può manipolare uno stato non direttamente esposto, manipolare più proprietà contemporaneamente oppure manipolare proprietà basate sulla logica interna. Invocare un'azione potrebbe anche far scattare un processo sulla Thing che manipola lo stato nel tempo. Esempi di azioni possono essere diminuire la luminosità di una luce nel tempo, oppure stampare un documento.
- **Eventi**: descrivono una fonte di eventi che invia dati in modo asincrono dalla Thing al consumatore. In questo caso non viene comunicato lo stato, ma *i cambiamenti di stato*. Gli eventi possono essere attivati da condizioni che non sono esposte come proprietà. Un esempio di evento può essere un allarme.

Binding Templates

I binding templates sono un meccanismo fondamentale nel WoT; essi permettono a un *WoT Consumer* di interagire, tramite la *Thing Description*, con *Thing* che presentano protocolli e payload diversi.
Il meccanismo di binding consente ai consumer di interagire con una varietà di Thing. Quando si descrive un dispositivo o una piattaforma IoT, il *binding template* corrispondente aiuta a recuperare i metadati di comunicazione da includere nella TD. Ogni *interaction affordance* deve avere un binding a un protocollo e un formato di payload: la *Thing Description* fornisce il *form* che contiene tutte le informazioni necessarie al consumer per eseguire un'operazione. Il consumer che elabora una TD implementa poi il binding template richiesto.
Esistono tre tipi principali di binding templates:

- **Protocol Binding Templates**: definiscono come i protocolli a livello *application* mappano i loro tipi di messaggio ai forms delle TD, specificano come i metodi esistenti (POST, GET, ...) possono essere usati in una TD per legarsi alle operazioni astratte (readproperty, invokeaction, ...), definiscono lo schema URI del protocollo.
- **Payload Binding Templates**: descrive i diversi formati di payload e tipi di media che possono essere rappresentati in una TD tramite i "forms".
- **Platform Binding Templates**: combinano l'uso di protocolli e payload in un modo specifico, come richiesto da piattaforme e framework IoT particolari. Sono dipendenti dai *Protocol e Payload Binding Templates*.

2.2.2 WoT Building Blocks

L'implementazione pratica di sistemi WoT, indipendentemente dall'architettura adottata, richiede la gestione di compiti comuni che possono essere considerati blocchi *fondamentali* generici di qualsiasi sistema WoT. Gli sviluppatori devono selezionare soluzioni

adeguate per rendere le Web Things accessibili, scopribili, interrogabili e sicure in un contesto di rete, supportando l'interoperabilità attraverso formalismi appropriati che consentano la comprensibilità automatica dlle capacità delle WT.
Sono stati individuati 6 building blocks[11]:

- **WT Modeling**: definisce uno schema concettuale per descrivere risorse e capacità offerte da una WT in modo uniforme e comprensibile dalle macchine. I principali standard includono *HSML, CoRAL, Web Thing Model e Thing Description.* La Thing Description (TD) rappresenta la soluzione più rilevante, poichè descrive una Thing in modo comprensibile sia per le macchine che per gli esseri umani, contenendo informazioni come nome, ID, descrizioni e metadata per le Interaction Affordances. Una TD può essere vista come il file *index.html* delle Things.

- **WT Annotation**: le tecnologie semantiche arricchiscono le WT con significato e abilitano l'interazione autonoma. La ricerca si divide in studi che propongono di arricchire il modello WT con ontologie specifiche del dominio e studi che sfruttano tali annotazioni per estrarre conoscenza dalle WT attraverso tecniche di *Machine Learning.* L'obiettivo è creare vocabolari che supportino le applicazioni composite in diversi ecosistemi WoT.

- **WT Access**: i meccanismi di accesso abilitano le WT a interagire sulla rete attraverso quattro approcci principali.Il *Direct Integration* utilizza dispositivi *IP-enabled* con server Web integrati, l'*Indirect Integration* sfrutta gateway intermediari per tradurre protocolli, il *Cloud Integration* si basa su piattaforme esterne per superare problemi di connettività, mentre il *Multi-pattern Integration* combina più approcci come nell'architettura W3C WoT.

- **WT Discovery**: la scoperta di dispositivi embedded connessi al Web rappresenta una sfida critica del WoT[15]. Il problema si divide in scoprire gli indirizzi di rete dei dispositivi e comprendere le loro funzionalità. Le soluzioni si raggruppano in tre categorie: *multi-layer* (processi distribuiti su più livelli architetturali), basate su *directory* (componenti registry con meccanismi di notifica), e basate su *servizi* (processi software autonomi per la scoperta decentralizzata).

- **WT Mashup**: le applicazioni Mashup raccolgono e gestiscono dati da WT eterogenee in modo *seamless.* Si distinguono in servizi repository che fungono da collettori di dati WoT (spesso integrati con Social Network per sfruttare meccanismi di fiducia) e toolkit di composizione che abilitano l'orchestrazione automatica delle WT utilizzando modelli uniformi come quelli del W3C WoT per estendere il concetto di TD alle applicazioni composite.

- **WT Seucring**: la sicurezza comprende autorizzazione, protezione, integrità e riservatezza dei dati WT. Il W3C WoT adotta un approccio descrittivo che supporta

i modelli di sicurezza esistenti anzichè introdurne di nuovi. Le principali sfide includono il provisioning sicuro dei metadata, la manutenzione degli aggiornamenti, e vulnerabilità specifiche come gestione di link locali, analisi delle vulnerabilità esposte dalla TD, scoperta sicura e sicurezza distribuita end-to-end.

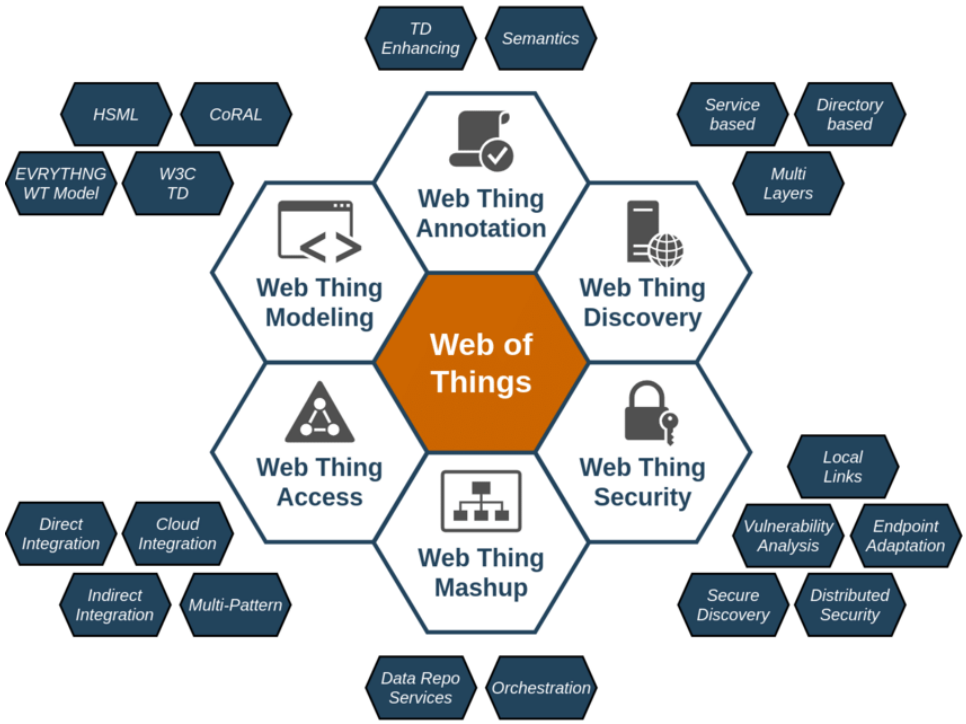

Figura 2.6: Building Blocks del WoT[11]

2.2.3 node-wot e kotlin-wot

Sono nati diversi framework che implementano la creazione di *WoT server e clients* in diversi linguaggi, sviluppati dal gruppo *Eclipse Thingweb*[16].
Il primo framework sviluppato è *node-wot*, per implementare server e client WoT in Node.js [17].
Di recente sono nati e continuano a nascare framework in vari linguaggi per WoT, come *pywot* per Python e quello che useremo per il progetto, ***kotlin-wot***, sviluppato per creare client e server WoT in ambito *kotlin*, del quale non si trovano ancora implementazioni Servient *mobile*.

Capitolo 3

Android WoT Servient

Questo capitolo parla del progetto *Android WoT Servient*, un *Servient WoT* realizzato per dispositivi Android basato sulle ultime specifiche del W3C usando il framework *kotlin-wot*. Non esiste allo stato attuale[18] un'implementazione completamente per Android.
Nelle seguenti sezioni vengono definiti obiettivi, requisiti tecnici e funzionali, parlando dei principali componenti che costituiscono l'architettura.

3.1 Obiettivo

Il progetto sviluppato è un *WoT Servient* per dispositivi Android sviluppato in Kotlin, conforme all'ultimo aggiornamento delle specifiche del W3C datato [13] 5 Dicembre 2023. Non avendo trovato implementazioni per kotlin-wot, il progetto prende spunto dalla libreria *node-wot*, rapportando il tutto al linguaggio Kotlin.
Obiettivo del progetto è creare un *Servient* (server e client) che giri completamente sullo smartphone, trasformando i dispositivi Android in nodi attivi dell'*Internet of Things*.
La diffusione dei dispositivi Android rappresenta un'opportunità unica per l'espansione dell'ecosistema *WoT*. Con oltre il 70% del mercato globale degli smartphone[4], Android offre una base installata enorme che può essere sfruttata per creare una rete distribuita di *Things* interconnessi, decentralizzando l'accesso alle tecnologie WoT senza richiedere hardware specializzato.
I moderni smartphone Android integrano una ricchezza di sensori senza precedenti, includendo ad esempio accelerometri, giroscopi, magnetometri, sensori di luminosità e pressione. Questa varietà di capacità sensoriali, combinata con fotocamere ad alta risoluzione e microfoni di qualità, trasforma ogni dispositivo in una stazione di monitoraggio ambientale completa, capace di raccogliere dati multidimensionali sull'ambiente circostante.
Il potenziale per applicazioni di *edge computing* e IoT mobile rappresenta un ulteriore

vantaggio strategico. I dispositivi Android moderni dispongono di processori potenti e connettività ubiqua (WiFi, 4G, 5G, Bluetooth), permettendo elaborazione locale dei dati e comunicazioni in tempo reale. Questo consente di implementare soluzioni IoT distribuite dove i dispositivi mobili fungono da sensori intelligenti in movimento.

3.2 Requisiti

Il progetto si basa sul framework *kotlin-wot* che fornisce le funzionalità base per implementare Servient WoT conformi alle specifiche W3C:

- gestione Thing Description,
- supporto per Interaction Affordances (properties, actions, events),
- binding protocols (HTTP, MQTT, WebSocket),
- runtime del Servient.

Inoltre, l'applicazione estende queste funzionalità del framework con requisiti specifici per l'ambiente mobile:

- **Esposizione dei sensori del dispositivo Android come Interaction Affordances**: deve essere possibile esporre i sensori del telefono, rendendoli *properties* del WoT.
- **Esposizione di foto e audio come Interaction Affordances**: deve essere possibile scattare foto e registrare audio, rendendo anch'essi *properties* WoT.
- **Creazione di una TD "smartphone"**: deve essere creata, all'avvio del server, una TD "smartphone" che contiene tutte le proprietà e le azioni possibili.
- **Selezione dinamica dei sensori**: deve essere possibile aggiungere/rimuovere i sensori a cui si può accedere tramite una pagina *Impostazioni*, gestendo la TD di conseguenza.
- **Invocazione *WoT Actions***: deve essere possibile, tramite ad esempio bottoni, invocare azioni WoT che comprendono i valori dei sensori.
- **Raccolta statistiche di utilizzo**: per monitorare le performance e l'utilizzo del server.

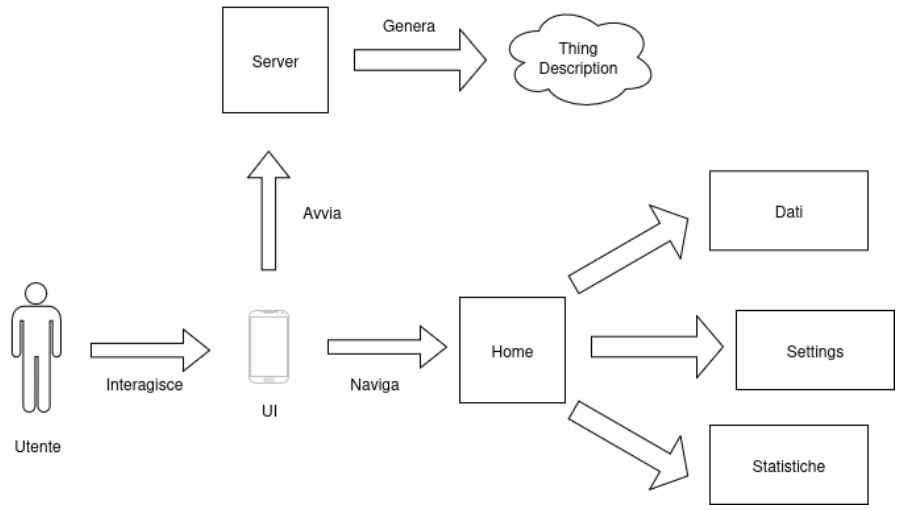

Figura 3.1: Architettura Android WoT Servient

3.3 Architettura

In questa sezione verrà analizzata l'architettura del progetto.
Il progetto è sostanzialmente composto da varie Activity/Fragment per visualizzazione dati e statistiche, un Service che mantiene attivo il server anche se l'applicazione non è in primo piano, e la classe Server che si occupa della creazione di server e TD.

3.3.1 Service

Il *Service* si occupa di avviare e mantenere attivo il server anche quando si cambia applicazione e la nostra non è più in primo piano.
Si occupa anche di gestire il cambio di preferenze nei sensori, stoppando e riavviando il server quando necessario. Infine, crea una notifica persistente dalla quale è possibile fermare il server.

3.3.2 Server

Il *Server* è la parte centrale del progetto. Qui viene creata la thing "Smartphone" e vengono aggiunte le varie proprietà in base ai sensori scelti.
Vengono inoltre create una serie di *actions* invocabili da una schermata, che si interfacciano con i valori dei sensori.

3.3.3 Activities

Le varie activities si occupano di interagire con l'utente, permettendogli di vedere i dati dei sensori in tempo reale, invocare le actions tramite dei bottoni, gestire la parte multimediale di foto e audio e gestire le impostazioni.

3.3.4 Gestione dati

Il sistema implementa componenti specializzati per la gestione centralizzata dei dati sensoriali e delle statistiche di utilizzo. Questi componenti si occupano della storicizzazione dei valori dei sensori, della persistenza delle configurazioni utente e della raccolta di metriche sulle interazioni con il servizio WoT.

3.3.5 Funzionamento App

L'utente interagisce con il Servient Android attraverso un'interfaccia grafica organizzata in quattro sezioni principali, accessibili tramite una barra di navigazione inferiore.
All'apertura dell'applicazione, l'utente viene accolto dalla sezione **Home** che fornisce una panoramica delle funzionalità e un pulsante per avviare il server WoT. Una volta avviato, il servizio rimane attivo in background tramite una notifica persistente, permettendo l'accesso ai sensori anche quando l'applicazione non è in primo piano.
Attraverso la sezione **Settings** l'utente può selezionare dinamicamente quali sensori del dispositivo esporre come proprietà WoT. Il sistema rileva automaticamente i sensori disponibili (accelerometro, giroscopio, magnetometro, sensori di luminosità e pressione) e permette di abilitarli o disabilitarli singolarmente. Ogni modifica viene applicata in tempo reale, aggiornando la *Thing Description*.
La sezione **Data** offre due modalità di interazione: visualizzazione dei dati sensoriali in tempo reale con possibilità di *refresh* automatico oppure manuale, e accesso alle funzionalità multimediali per gestione di foto e registrazione audio. I dati vengono presentati attraverso interfacce dinamiche che si adattano ai sensori attualmente attivi.
L'applicazione include inoltre strumenti per testare le azioni computazionali implementate al server, permettendo di invocare funzioni come calcolo della magnitudine dell'accelerazione, determinazione dell'orientamento o verifica se il dispositivo è in tasca. Ogni azione viene abilitata solo se i sensori prerequisiti sono attivi.
La sezione **Stats** presenta grafici interattivi che mostrano l'utilizzo del sistema, inclusa la distribuzione degli accessi ai diversi sensori, i tipi di interazione più frequenti e l'evoluzione temporale dei dati sensoriali.

Capitolo 4

Implementazione

Per la realizzazione del progetto è stato necessario usare diverse tecnologie e componenti. Di seguito saranno descritte la struttura del codice e le tecnologie utilizzate. Il progetto è *open source* ed è possibile raggiungere la *repository* tramite questo link: *https://github.com/UniBO-PRISMLab/AndroidWoTServient*.

4.1 Struttura e dettagli del codice

Il progetto è implementato come una singola applicazione Android che integra tutte le funzionalità necessarie per trasformare lo smartphone in una Thing WoT. L'architettura dell'applicazione è organizzata in un unico modulo principale che combina la logica del server WoT, la gestione dei sensori, la pubblicazione degli eventi e l'interfaccia utente in un sistema coeso e integrato.
L'applicazione è strutturata seguendo le *best practice* di sviluppo Android, con una chiara separazione delle responsabilità tra i diversi componenti. Data la natura monolitica ma ben organizzata del progetto, ogni componente è stato progettato per svolgere un ruolo specifico nell'ecosistema WoT, dalla gestione dei sensori fisici alla pubblicazione degli eventi in tempo reale attraverso diversi protocolli (HTTP, MQTT, WebSocket). La struttura modulare del codice facilita la manutenzione e l'estensione delle funzionalità, pur mantenendo tutte le componenti all'interno di una singola applicazione.
Vediamo le classi principali.

4.1.1 WoTService.kt

Una delle classi principali del progetto, implementa un servizio in foreground che gestisce il ciclo di vita di un server WoT sul dispositivo. Responsabile di avviare, fermare e aggiornare il server. All'avvio crea una notifica per mantenersi attivo in background, assicurando che Android non lo termini arbitrariamente. Usa un Mutex per garantire che

le operazioni di start e stop non si sovrappongano tra loro. Configura HTTP, WebSocket e MQTT in base alle preferenze utente.
La funzione *start* legge le preferenze delle utente (porta, hostname, IP locale, configurazioni MQTT e WebSocket) per configurare l'indirizzo del server HTTP, crea e configura il servient con i protocolli abilitati, crea un'istanza di Wot, avvia il server e crea un'istanza di Server, poi aggiorna lo stato interno e salva nelle preferenze che il server è partito.
La funzione *stop* invece ferma server e servient, pulisce riferimenti e flag interni e aggiorna le preferenze mandando un broadcast di stato.
La funzione *udpate* invece, attraverso un *BroadcastReceiver* ascolta gli eventi di aggiornamento delle preferenze (*PREFERENCES_ UPDATED*) e reagisce di conseguenza: per cambiamenti di porta/hostname ferma il server e lo riavvia, mentre per i cambiamenti dei sensori aggiorna dinamicamente la Thing senza effettuare uno stop completo quando possibile.
I protocolli configurati sono: **HTTP** sempre abilitato, **WebSocket** abilitato di default sulla porta 8081 con binding configurabile, **MQTT** opzionale, configurabile tramite broker host e porta.

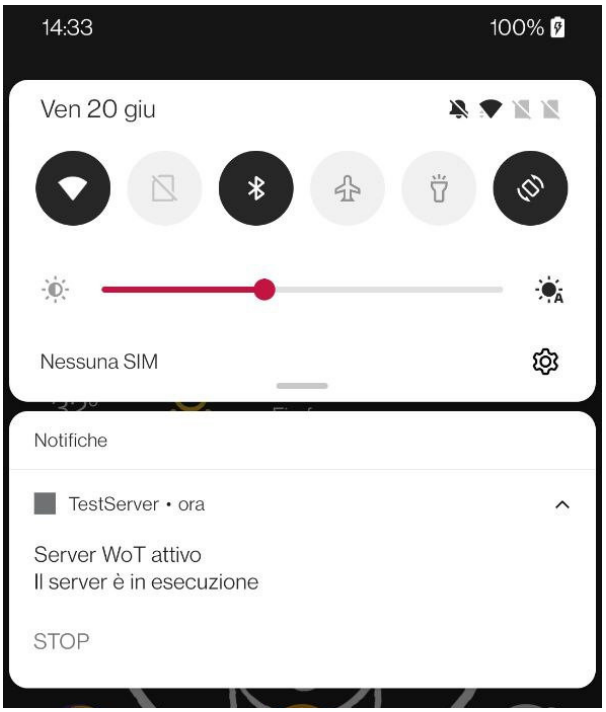

Figura 4.1: Notifica del Server attivo

4.1.2 Server.kt

Questa invece è la classe che implementa la logica di creazione della *Thing* "smartphone" che rappresenta i sensori selezionati e le funzionalità multimediali come foto e audio.
La classe Server inizialmente si occupa di leggere le preferenze dell'utente per decidere quali sensori abilitare e configurare l'indirizzo e la porta su cui esporre il server, poi crea dinamicamente la Thing con proprietà osservabili che rappresentano i valori in tempo reale dei sensori abilitati, espone le proprietà per la gestione di foto e audio, con anche azioni per scattare foto e registrare audio, aggiornare le proprietà "audio" e "photo" con dati codificati in Base64. Si occupa anche di fornire una serie di azioni utili basate sui dati dei sensori come calcolare la magnitudine, determinare la direzione del Nord magnetico, orientamento, inclinazione e verifica se il dispositivo è in tasca.
Gestisce la lettura dei valori sensoriali in modo asincrono tramite handler di lettura delle proprietà che leggono direttamente i dati dei sensori Android, gestisce le azioni WoT con handler che attivano funzionalità multimediali, aggiornano le proprietà con i dati ottenuti e salvano i file multimediali sul dispositivo.
Crea e gestisce un'istanza di *SensorPublisher* per la pubblicazione automatica degli eventi dei sensori, necessaria per i test tramite MQTT e WebSocket.
Infine traccia anche le richieste di lettura e invocazione delle azioni per scopi statistici o di monitoraggio.

```
stringProperty("photo") {
    title = "Last captured photo"
    description = "Base64 encoded image"
    readOnly = true
    observable = false
}
action<Unit, Unit>("takePhoto") {
    title = "Capture a new photo"
    description = "Takes a new photo and updates photo property"
}
```

Figura 4.2: Esempio di dichiarazione Action

```
setPropertyReadHandler("photo") {
    ServientStats.logRequest(thingId, "readProperty", "photo")
    InteractionInput.Value(jsonNodeFactory.textNode(currentPhotoBase64))
}
setActionHandler("takePhoto") { _: WoTInteractionOutput, _: InteractionOptions? ->
    ServientStats.logRequest(thingId, "invokeAction", "takePhoto")
    MediaUtils.takePhoto(context)
    InteractionInput.Value(jsonNodeFactory.nullNode())
}
```

Figura 4.3: Esempio implementazione Action

4.1.3 SensorPublisher

Classe specializzata nella pubblicazione automatica e continua dei valori dei sensori attraverso eventi WoT. Utilizza *coroutine* per pubblicare i valori dei sensori ogni 500 ms senza bloccare il thread principale.
Gli eventi pubblicati vengono automaticamente inoltrati attraverso tutti i protocolli abilitati (HTTP, WebSocket e MQTT) tramite il meccanismo *emitPropertyChange* del framework WoT.

4.1.4 MainActivity

Rappresenta l'*Activity* principale dell'applicazione e funge da contenitore per tutti i *Fragment* che compongono l'interfaccia utente. Implementa un'architettura basata su navigazione *bottom-tab* che permette agli utenti di spostarsi facilmente tra le diverse sezioni dell'applicazione.
La classe gestisce il ciclo di vita dell'interfaccia utente principale e coordina la visualizzazione del Fragment attraverso un *FrameLayout* centrale. Al primo avvio carica automaticamente l'*HomeFragment* come schermata predefinita, mentre successivamente gestisce le transizione tra i viersi Fragment in base alla selezione dell'utente nella barra di navigazione inferiore.
La navigazione è implementata tramite un *BottomNavigationView* che permette l'accesso rapido alle quattro sezioni principali: **Home** per la panoramica generale, **Stats** per le statistiche e i grafici, **Data** per la visualizzazione dei dati sensoriali e **Settings** per la configurazione del sistema. Ogni cambio di tab sostituisce completamente il Fragment corrente, garantendo un'esperienza utente fluida e consistente con le linee guida *Material Design* di Android[19].

4.1.5 HomeFragment

Rappresenta la schermata principale dell'applicazione, fornendo un punto di accesso centralizzato per avviare il server WoT e una panoramica delle funzionalità disponibili.
Il Fragment gestisce automaticamente le richieste dei permessi necessari per il funzionamento del servizio, in particolare il permesso per le notifiche, assicurando che tutti i prerequisiti siano soddisfatti prima dell'avvio del servizio. Presenta un testo descrittivo sulle capacità dell'applicazione e fornisce un pulsante di avvio prominente per iniziare l'esperienza WoT.
Al momento dell'avvio del servizio, gestisce la creazione del servizio in foreground tramite *startForegroundService*, garantendo che il server WoT rimanga attivo anche quando l'applicazione non è in primo piano. Fornisce anche feedback immediato all'utente tramite *Toast* informativi e gestisce accuratamente i casi di errore nella richiesta dei permessi.

4.1.6 SensorDataFragment

Rappresenta il componente centrale per la visualizzazione in tempo reale dei dati provenienti dai sensori del dispositivo, implementando un client WoT completo che si connette al Thing "smartphone" locale per recuperare e presentare i valori sensoriali. Fornisce un'interfaccia dinamica che si adatta automaticamente ai sensori attualmente abilitati dall'utente.
Il Fragment implementa un sistema di connessione robusto che attende la disponibilità del server prima di tentare la connessione, usando *MultiValueSensorClient* per gestire

le comunicazioni col Thing. Durante l'inizializzazione, interroga le preferenze utente per determinare quali sensori sono attivi e crea dinamicamente l'interfaccia utente con *TextView* appropriati per ciascuna proprietà sensoriale, distinguendo tra sensori a valore singolo e multiplo.
La gestione degli aggiornamenti è implementata attraverso due modalità: aggiornamento manuale tramite pulsante e aggiornamento automatico configurabile tramite switch. La modalità automatica utilizza *coroutine* con intervalli di 500ms per fornire una visualizzazione real-time dei dati sensoriali. Inoltre il Fragment integra con *SensorDataHolder* per persistere i dati raccolti, alimentando il sistema di statistiche e grafici dell'applicazione. L'interfaccia include inoltre un pulsante di navigazione verso *SensorActionsFragment*.

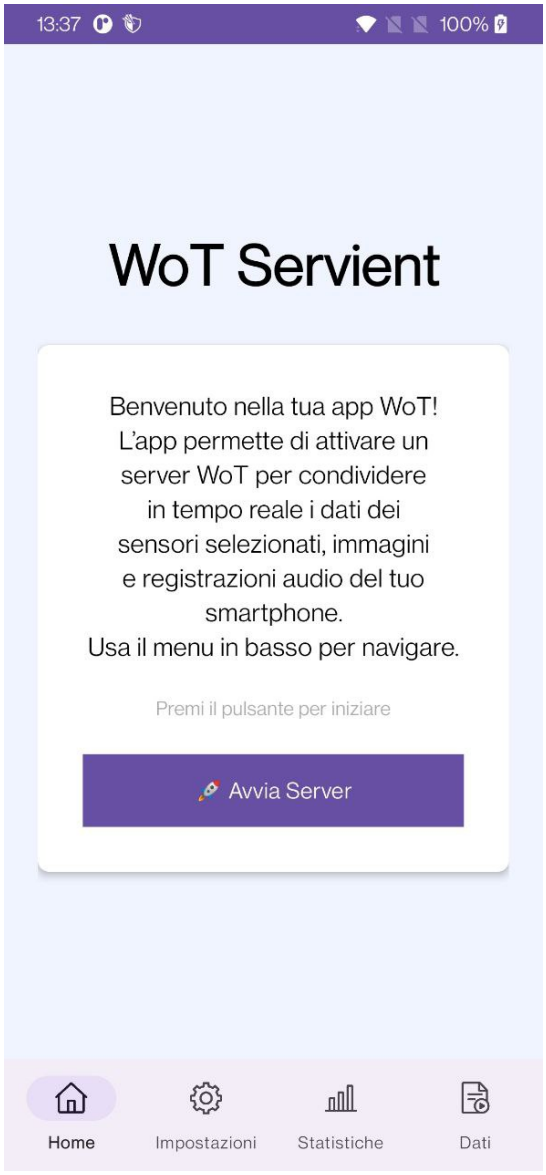

Figura 4.4: Schermata Home

Figura 4.5: Schermata SensorData

4.1.7 SensorActionsFragment

Implementa un'interfaccia di testing per le azioni computazionali del Thing WoT, permettendo agli utenti di invocare e testare le funzionalità implementate dal server. Fornisce un ambiente controllato per verificare il corretto funzionamento delle azioni che trasformano i dati sensoriali grezzi in informazioni di alto livello utili per applicazioni

specifiche.
Il Fragment gestisce cinque azioni principali: calcolo della magnitudine dell'accelerazione, determinazione della direzione del Nord magnetico, calcolo dell'orientamento completo, stima dell'inclinazione del dispositivo e verifica se il telefono è in tasca. Ogni azione viene abilitata dinamicamente in base ai sensori attualmente attivi, implementando controlli di prerequisiti che verificano la disponibilità dei sensori necessari prima di permetterne l'invocazione.
La classe utilizza un *BroadcastReceiver* per reagire ai cambiamenti di configurazione dei sensori, aggiornando automaticamente la disponibilità delle azioni quando l'utente modifica le preferenze. L'integrazione con le *coroutine* garantisce che le chiamate WoT non blocchino l'interfaccia utente, fornendo un'esperienza fluida anche durante operazioni computazionalmente intensive.

4.1.8 DynamicSensorSettingsFragment

Implementa un'interfaccia dinamica per la configurazione dei sensori disponibili sul dispositivo, sostituendo l'approccio statico con una generazione automatica delle preferenze basata sui sensori effettivamente presenti. Estende *PreferenceFragmentCompat* per fornire un'interfaccia standardizzata che si adatta automaticamente alle capacità hardware del dispositivo specifico.
Il Fragment interroga il SensorManager per ottenere la lista dei sensori supportati, filtrando solo quelli compatibili con il framework WoT e applicando una logica di selezione che privilegia i sensori *non-wakeup*, al fine di evitare sensori doppioni. Per ogni sensore disponibile crea dinamicamente una *SwitchPreferenceCompat* con nome *user friendly*, consentendo agli utenti di abilitare o disabilitare selettivamente i sensori da esporre.
Implementa un sistema di notifiche intelligenti che distingue tra modifiche minori e modifiche significative che richiedono il riavvio del server. Durante il normale utilizzo invia broadcast "*sensors*" per aggiornamenti dinamici, mentre al termine della sessione di configurazione valuta se le modifiche richiedono un riavvio completo, inviando in tal caso un broadcast "*sensors_restart*" per garantire la coerenza del sistema.

Figura 4.6: Schermata Actions

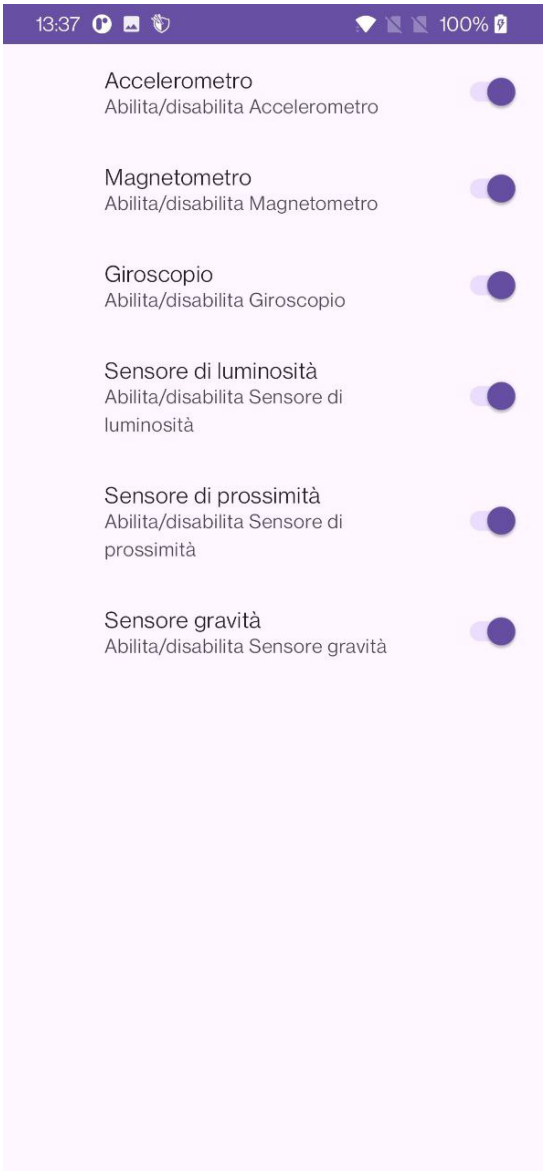

Figura 4.7: Selezione dei sensori

4.1.9 ServientStats

Classe *singleton* che implementa un sistema completo di raccolta e gestione delle statistiche di utilizzo del server WoT. Traccia metriche dettagliate su tutte le interazioni con i Thing esposti, inclusi conteggi per tipo di interazione, accessi per Thing specifico e utilizzo delle singole affordances.
La classe mantiene contatori per il numero totale di richieste, richieste per ciascuna Thing, tipi di interazione (readProperty o invokeAction) e accessi specifici alle affordance individuali e calcola automaticamente l'uptime del server basandosi sul timestamp di avvio.
Implementa persistenza automatica delle statistiche attraverso integrazione con *ServientStatsPrefs*, salvando i dati dopo ogni richiesta per garantire che le informazioni non vadano perse in caso di chiusura inaspettata dell'applicazione. Ha inoltre una funzione di reset che permette di azzerare tutte le statistiche, utile per iniziare nuove sessioni di monitoraggio.

4.1.10 SensorDataHolder

Classe *singleton* responsabile della gestione centralizzata e dello storico dei dati provenienti dai sensori del dispositivo. Mantiene una cache in memoria dei valori sensoriali più recenti per ogni proprietà attiva, limitando automaticamente la dimensione dello storico per ottimizzare l'uso della memoria.
La classe implementa un sistema di notifica automatica per i cambiamenti nelle proprietà disponibili, inviando broadcast quando nuovi sensori vengono attivati o disattivati. Mantiene uno storico limitato a 100 valori per proprietà, eliminando automaticamente i dati più vecchi quando il limite viene raggiunto, garantendo prestazioni costanti anche durante lunghi periodi di funzionamento.
Fornisce API per l'accesso ai dati storici, valori più recenti e informazioni aggregate come il conteggio dei punti dati per proprietà. La gestione centralizzata dei dati sensoriali facilita l'integrazione con i componenti di visualizzazione e egarantisce coerenza nei dati presentati attraverso diverse parti dell'applicazione. Il sistema di broadcast permette aggiornamenti reattivi dell'interfaccia utente quando la configurazione dei sensori cambia dinamicamente.

4.1.11 StatsFragment

Rappresenta il componente dedicato alla visualizzazione delle statistiche di utilizzo del server WoT, fornendo un'interfaccia completa per monitorare le performance e l'attività del sistema attraverso grafici interattivi e metriche aggregate.
Il Fragment implementa tre tipologie di visualizzazione dati utilizzando la libreria *MPAndroidChart*. Il grafico **a torta** mostra la distribuzione degli accessi alle diverse affordance del dispositivo, permettendo di identificare rapidamente quali sensori vengono utilizzati maggiormente. I dati vengono presentati con nome *user-friendly* ottenuti tramite la funzione *getFriendlyNameProperty()* che converte i nomi tecnici dei sensori in etichette comprensibili (es: "TYPE_ACCELEROMETER" diventa "Accelerometro").
Il grafico **a barre** visualizza i tipi di interazione registrati dal sistema, distinguendo tra operazioni di lettura proprietà (*readProperty*) e invocazione delle azioni (*invokeAction*). Questa visualizzazione facilita la comprensione del pattern di utilizzo e delle modalità di interazione più frequenti con il Thing WoT. Il grafico (**a linee**) presenta l'evoluzione temporale dei valori sensoriali selezionati tramite uno Spinner dinamico. L'utente può selezionare qualsiasi sensore attivo per visualizzare lo storico dei valori, con aggiornamenti automatici ogni 5 secondi. Il grafico utilizza interpolazione per una visualizzazione fluida delle curve dei dati.
La gestione dinamica dell'interfaccia è implementata attraverso un *BroadcastReceiver* che ascolta gli eventi di aggiornamento delle preferenze, reagendo automaticamente quando l'utente modifica la configurazione dei sensori. Lo Spinner viene aggiornato dinamicamente per riflettere solo i sensori attualmente attivi, mantenendo la selezione precedente quando possibile.
Il Fragment integra completamente con la classe *ServientStats* per accedere alle metriche di utilizzo e con *SensorDataHolder* per ottenere i dati storici dei sensori. Include inoltre funzionalità di reset delle statistiche tramite un pulsante dedicato, permettendo di iniziare nuove sessioni di monitoraggio.

Figura 4.8: Uptime + tasto reset

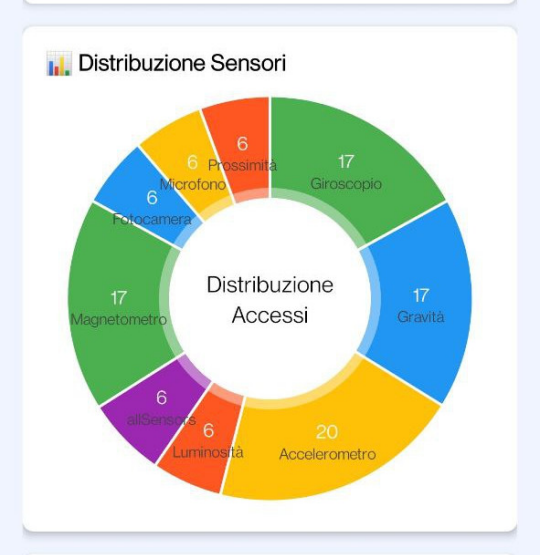

Figura 4.9: Grafico a torta

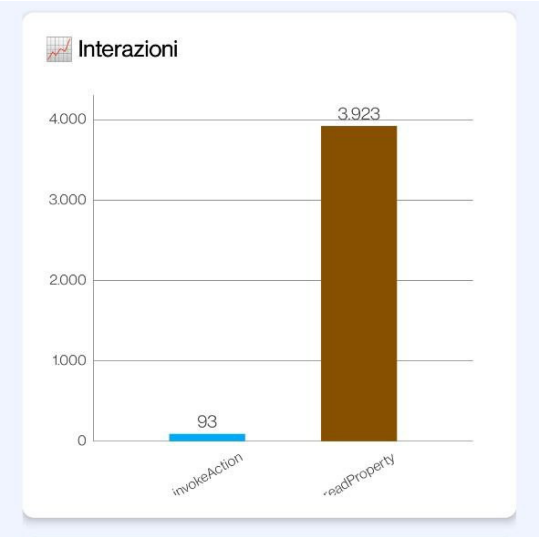

Figura 4.10: Grafico a barre

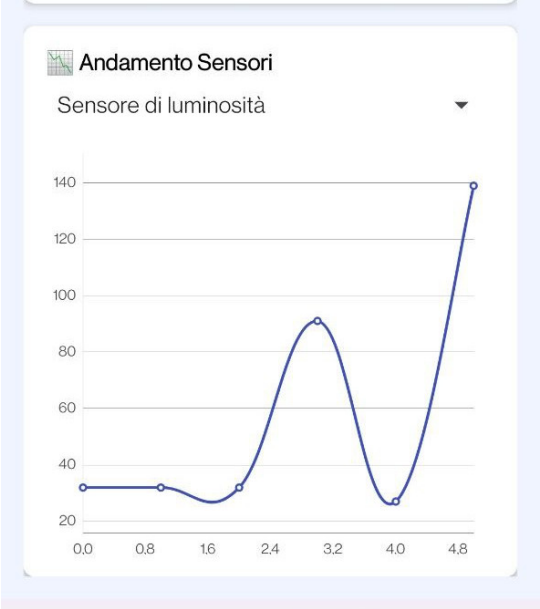

Figura 4.11: Grafico a linee

4.1.12 PicAudioActivity

Implementa l'interfaccia per la gestione delle funzionalità multimediali del WoT Servient, permettendo agli utenti di scattare foto e registrare audio esponendo queste capacità come *Interaction Affordances* del Thing "smartphone".
L'activity gestisce completamente il ciclo di vita delle operazioni multimediali, dalla richiesta dei permessi necessari (*CAMERA, RECORD_AUDIO, WRITE_EXTERNAL_STORAGE, READ_MEDIA_AUDIO*) all'esecuzione delle azioni WoT corrispondenti. Implementa un sistema robusto di gestione dei permessi che si adatta automaticamente alle diverse versioni di Android, richiedendo *READ_MEDIA_AUDIO* per Android 13+ e *WRITE_EXTERNAL_STORAGE* per versioni precedenti.
La **gestione fotografica** utilizza l'Intent *MediaStore.ACTION_IMAGE_CAPTURE* per attivare l'app fotocamera nativa del dispositivo. Una volta scattata la foto, l'immagine viene salvata in cache, convertita in formato Base64 e inviata al server WoT tramite l'azione *updatePhoto*. La proprietà "photo" viene quindi aggiornata e l'immagine viene visualizzata nell'*ImageView* dell'interfaccia con una *readProperty*.
La **gestione audio** implementa un workflow completo di registrazione attraverso le azioni WoT *startRecording* e *stopRecording*. Durante la registrazione, l'interfaccia viene aggiornata dinamicamente abilitando/disabilitando i pulsanti appropriati per guidare l'utente. Al termine della registrazione, l'audio viene automaticamente convertito in Base64 e reso disponibile tramite la proprietà "audio" del Thing.
L'activity funziona come client WoT completo, utilizzando la libreria *kotlin-wot* per connettersi al Thing "smartphone" locale. Implementa un sistema di connessione intelligente che attende la disponibilità del server prima di tentare la connessione.
La classe integra con *MediaUtils* per la gestione di basso livello delle operazioni multimediali e utilizza coroutine per garantire che tutte le operazioni di rete e I/O non blocchino l'interfaccia utente. Il sistema di gestione del ciclo di vita assicura la corretta pulizia delle risorse quando l'Activity viene distrutta.

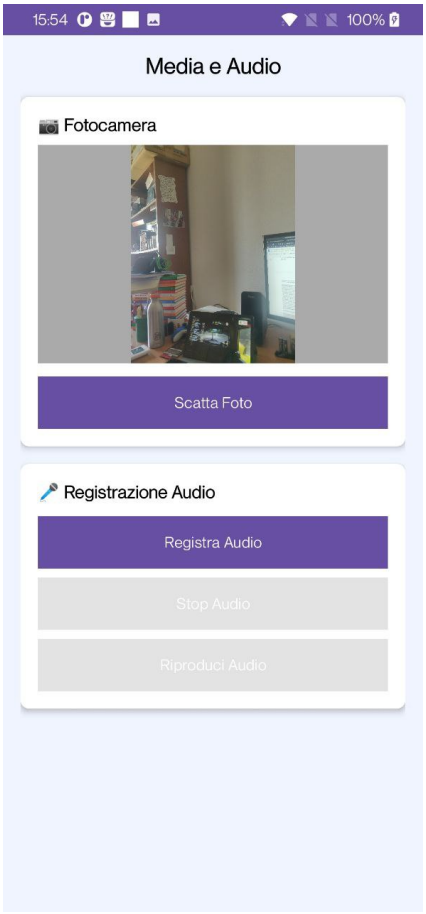

Figura 4.12: Schermata Media

4.2 Tecnologie utilizzate

4.2.1 kotlin-wot

La libreria principale utilizzata, basata sulla *repository* ufficiale di *Eclipse Thingweb* [20]. Questa libreria consente di creare e gestire *Things* digitali che rappresentano dispositivi fisici o servizi, esponendo le loro funzionalità attraverso interfacce web standardizzate.
L'architettura di *kotlin-wot* si basa su diversi componenti fondamentali che collaborano per fornire una piattaforma completa per lo sviluppo di applicazioni IoT. Il *Servient* rappresenta il componente principale che gestisce sia la produzione che il consumo di Things, fungendo da ponte tra il mondo fisico dei sensori e l'ambiente digitale delle applicazioni web. Il componente *Wot* fornisce le API di alto livello per creare e interagire con le Things, mentre i *binding protocol* come *HttpProtocolServer e HttpProtocolClientFactory* si occupano delle comunicazioni di rete.
La *Thing Description* costituisce il cuore del sistema, rappresentando un documento JSON-LD che descrive completamente le capacità di un dispositivo, incuse le proprietà leggibili, le azioni eseguibili e gli eventi generabili. Questo approccio basato su metadati consente una scoperta e integrazione automatica dei dispositivi, facilitando la creazione di ecosistemi IoT interoperabili.
Nel contesto di questa applicazione, *kotlin-wot* viene utilizzato per trasformare lo smartphone in un *Thing WoT* completo, esponendo i suoi sensori e le funzionalità attraverso API HTTP, MQTT e WebSocket. La classe *Server.kt* gestisce la creazione e configurazione del Thing principale denominato "smartphone", che aggrega tutte le affordances dei sensori abilitati dall'utente.
L'adozione dello standard WoT consente alla Thing smartphone di essere scoperto e utilizzato da qualsisasi client compatibile WoT, indipendentemente dalla tecnologia di implementazione. La standardizzazione delle interfacce facilita l'integrazione con altri sistemi e la creazione di applicazioni composite che utilizzano multiple sorgenti di dati.

4.2.2 MPAndroidChart

MPAndroidChart è una libreria *open source* per Android che fornisce componenti grafici avanzati per la visualizzazione dei dati. Sviluppata da Philipp Jahoda [21], questa libreria offre un'ampia gamma di tipologie di grafici personalizzati e performanti, rendendola una scelta ideale per applicazioni che necessitano di presentare dati in forma visiva.
La libreria si distingue per diverse caratteristiche che la rendono particolarmente adatta allo sviluppo di applicazioni Android. La varietà di grafici supportati include numerose tipologie di visualizzazioni, tra cui grafici a linee, a barre, a torta, radar, scatter plot e candlestick, con ogni tipologia ottimizzata per specifici casi d'uso. Le performance sono ottimizzate attraverso l'utilizzo di tecniche di rendering *hardware-accelerated* e l'implementazione di strategie specifiche per gestire in modo efficiente grandi dataset,

mantenendo fluidità anche con migliaia di punti di dati. La libreria offre inoltre un controllo granulare su ogni aspetto visivo dei grafici, inclusi colori, animazioni, etichette, legenda e assi, permettendo di creare visualizzazioni che si integrano perfettamente con il design dell'applicazione.

Implementazione nel progetto

Nel contesto di questa applicazione, *MPAndroidChart* è stata utilizzata per implementare tre diverse tipologie di visualizzazione dati nella sezione statistiche. Il grafico a torta viene utilizzato per mostrare la *distribuzione degli accessi* alle diverse affordance del dispositivo. Questo tipo di visualizzazione è perfetto per rappresentare proporzioni e percentuali, permettendo di identificare rapidamente quali sensori o funzionalità vengono utilizzati maggiormente.
Il grafico a barre è impiegato per visualizzare i tipi di interazione registrati dal sistema. La rappresentazione a barre facilita il confronto tra diverse categorie di dati, rendendo immediata la comprensione delle modalità di utilizzo più frequenti. Infine il grafico a linee viene utilizzato per mostrare l'evoluzione temporale dei dati dei sensori selezionati, risultando ideale per identificare trend, pattern e variazioni nel tempo dei valori rilevati dai sensori del dispositivo.

Vantaggi nell'utilizzo

L'adozione di *MPAndroidChart* ha portato diversi benefici significativi al progetto. La libreria ha permesso di implementare rapidamente visualizzazioni professionali senza dover sviluppare componenti grafici da zero, riducendo considerevolmente i tempi di sviluppo. Le animazioni integrate migliorano l'esperienza utente rendendo le transizioni tra diversi stati dei dati più fluide, e inoltre la capacità di gestire aggiornamenti in tempo reale dei grafici si è rivelata fondamentale per monitorare continuamente l'attività del sistema e l'utilizzo dei sensori.

4.2.3 Jackson

Jackson è una suite di librerie Java ad alte prestazioni per l'elaborazione di dati JSON, ampiamente considerata lo standard de facto per la serializzazione e deserializzazione JSON nell'ecosistema Java e Kotlin. La libreria si distingue per la sua velocità, flessibilità e completezza funzionale, offrendo un supporto completo per le specifiche JSON più avanzate.
Nell'applicazione, Jackson viene utilizzato per gestire la comunicazione tra il Thing WoT e i suoi client, convertendo i dati dei sensori in formati JSON standardizzati compatibili con le specifiche WoT. La classe *Server.kt* utilizza *JsonNodeFactory* per creare dinamicamente le risposte JSON appropriate per ciascun tipo di sensore e azione.

Per le proprietà sensoriali semplici, Jackson converte i valori float dei sensori in *NumberNode*, mentre per dati più complessi come l'orientamento del dispositivo vengono utilizzati *ArrayNode*.
L'uso di questa libreria porta diversi vantaggi. Le prestazioni ottimizzate della libreria garantiscono una latenza minima nella conversione dei dati sensoriali, aspetto cruciale per applicazioni real-time che richiedono aggiornamenti frequenti dei valori. La robustezza nella gestione degli errori e la capacità di gestire dati malformati contribuiscono alla stabilità complessiva del sistema.

4.2.4 Protocolli di comunicazione

Il framework kotlin-wot supporta multiple tecnologie di binding per garantire flessibilità nella comunicazione con le Things. L'implementazione corrente integra tre protocolli principali: **HTTP**, **MQTT** e **WebSocket**, ognuno ottimizzato per specifici scenari d'uso.
HTTP rappresenta il protocollo principale, utilizzando *HttpProtocolServer* per l'esposizione delle Things e *HttpProtocolClientFactory* per il consumo. HTTP garantisce compatibilità universale e semplicità di debugging attraverso strumenti standard del web.
MQTT viene implementato attraverso *MqttProtocolServer* e *MqttProtocolClientFactory*, configurabile tramite le preferenze utente per broker esterni come *test.mosquitto.org*. Questo protocollo risulta ideale per scenari IoT che richiedono comunicazione *publish/subscribe* efficiente e supporto per dispositivi con risorse limitate. La classe *SensorPublisher* utilizza MQTT per pubblicare automaticamente le variazioni dei valori sensoriali, creando messaggi JSON WoT completi con metadati temporali.
WebSocket completa l'ecosistema fornendo comunicazione *full-duplex* in tempo reale, implementato tramite *WebSocketProtocolServer*. Particolarmente utile per applicazioni che necessitano di aggiornamenti immediati dei dati sensoriali, WebSocket permette comunicazione bidirezionale efficiente mantenendo connessioni persistenti.

4.2.5 Gestione dei sensori Android

L'integrazione con il *Sensor Framework* di Android avviene attraverso una strategia multilivello che ottimizza prestazioni e affidabilità. Il *SensorManager* fornisce accesso unificato ai sensori hardware e software del dispositivo, mentre la logica di filtraggio privilegia sensori non-wakeup per minimizzare il consumo energetico ed evitare duplicazioni.
La lettura sincrona dei valori sensoriali è implementata attraverso *SensorEventListener* con *CountDownLatch* per garantire dati aggiornati durante le richieste WoT. Per la pubblicazione automatica, la classe *SensorPublisher* mantiene listener persistenti che emettono eventi WoT ogni 500ms, trasformando i dati Android in formato JSON standardizzato con metadati completi inclusi timestamp e identificatori univoci.

La mappatura tra sensori fisici e proprietà WoT gestisce automaticamente sensori a valore singolo (luminosità, pressione) e multi-asse (accelerometro, giroscopio), creando proprietà separate per ogni asse con nomenclatura standardizzata e unità di misura appropriate.

4.2.6 Gestione delle preferenze e configurazione

Il sistema di configurazione si basa su *SharedPreferences* Android per persistere impostazioni utenti e parametri di sistema. La configurazione dinamica dei protocolli permette di abilitare/disabilitare MQTT e WebSocket, configurare porte, hostname e credenziali broker.
Il pattern *BroadcastReceiver* implementato in WoTService gestisce aggiornamenti real-time delle preferenze, distinguendo tra modifiche che richiedono riavvio completo e quelle gestibili dinamicamente. Questo approccio garantisce responsività del sistema minimizzando interruzioni del servizio.

Capitolo 5

Validazione

Per validare l'efficacia e le performance del *Servient WoT Android* sviluppato, è stata condotta un'analisi comparativa tra il server WoT implementato e un server tradizionale. L'obiettivo è quantificare l'overhead introdotto dalle specifiche WoT e dimostrare che tale overhead è giustificato dai benefici in termini di standardizzazione e interoperabilità.

5.1 Metodologia di Test

È stata sviluppata un'applicazione Android dedicata (*PerformanceTester*) per condurre test automatizzati e sistematici. L'applicazione implementa un framework di testing completo che supporta **HTTP** tramite richieste REST tradizionali e gli endpoint WoT, **MQTT** tramite pubblicazione/sottoscrizione su broker pubblici (*test.mosquitto.org*) e **WebSocket**, tramite comunicazione real-time bidirezionale.

Per garantire un confronto equo, è stato implementato un server Android tradizionale (app *ServerRequests*), che espone le stesse funzionalità del WoT Servient ma senza l'overhead delle specifiche W3C.

Ogni sessione di test comprende:

- **500 richieste per protocollo** per server normale e WoT.
- **Misurazione di**: tempo di risposta, dimensione pacchetto, tasso di successo.
- **Raccolta di statistiche** aggregate per analisi comparative.
- **Ambiente controllato**: stesso dispositivo, stessa rete WiFi.

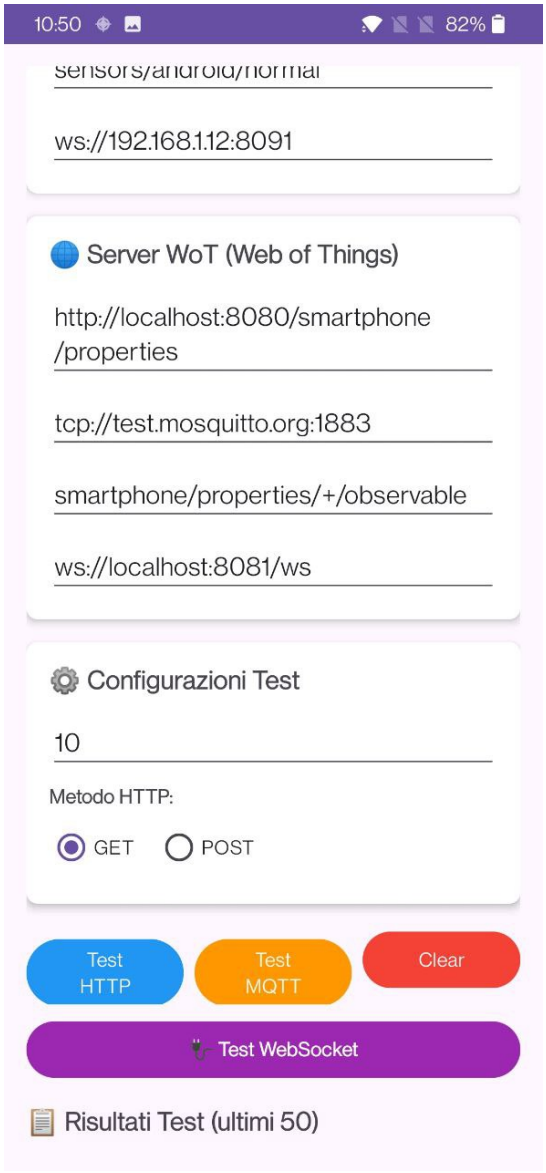

Figura 5.1: App PerformanceTester

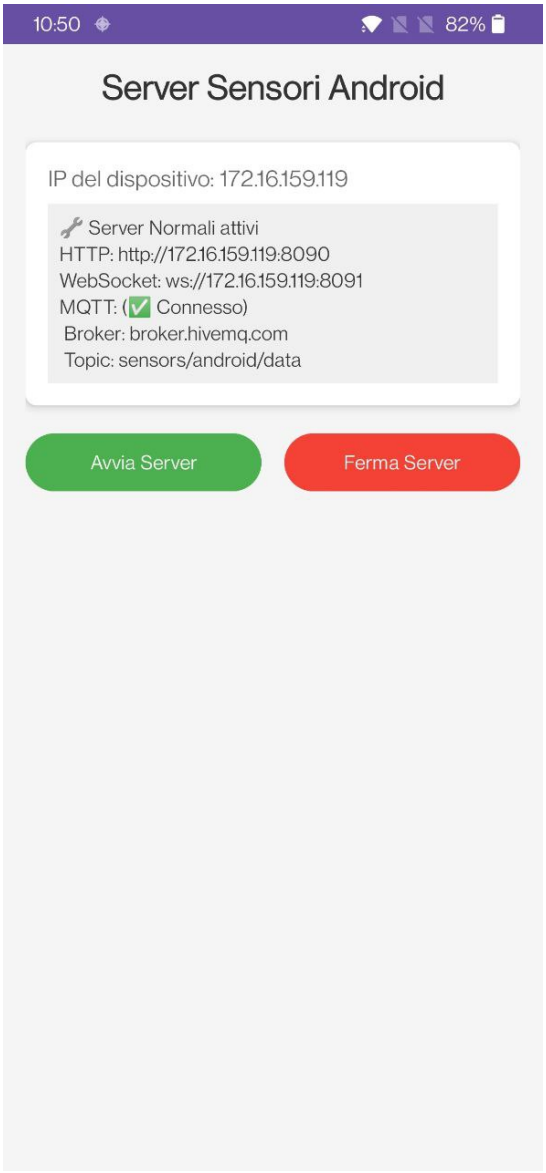

Figura 5.2: App ServerRequests

I test sono stati condotti su un dispositivo **Oneplus 6T** in rete **WiFi**, eseguendo 500 richieste per ogni combinazione protocollo-server per un totale di 3000 test. I protocolli testati comprendono richieste GET HTTP per tutti i sensori, sottoscrizione MQTT con timeout di 15 secondi, e comunicazione WebSocket per un singolo sensore con timeout di 10 secondi.

5.2 Performance

Metrica	Server Tradizionale	Server WoT
Richieste totali	500	500
Tasso di successo	100%	100%
Tempo di risposta medio	7.32 ms	322.73 ms
Tempo di risposta min/max	2-67 ms	190-458 ms
Dimensione pacchetto medio	290 bytes	654 bytes
Dimensione pacchetto min/max	275-297 bytes	642-664 bytes

Tabella 5.1: Performance HTTP: confronto server tradizionale e WoT

Analisi overhead HTTP

- **Overhead tempo di risposta**: +315 ms
- **Overhead dimensione pacchetto**: +365 bytes

L'incremento è dovuto principalmente al processing delle Thing Description e alla gestione delle Interaction Affordances secondo le specifiche W3C WoT.

Metrica	Server Tradizionale	Server WoT
Richieste totali	500	500
Tasso di successo	89%	92%
Tempo di risposta medio	15 s	13.5 s
Tempo di risposta min/max	3-76 s	1.7-41 s
Dimensione pacchetto medio	7.7 bytes	108 bytes
Dimensione pacchetto min/max	0-13 bytes	0-127 bytes

Tabella 5.2: Performance MQTT: confronto server tradizionale e WoT

Analisi overhead MQTT

- **Overhead tempo di risposta**: - 1569 ms
- **Overhead dimensione pacchetto**: +100 bytes

Il protocollo MQTT presenta variabilità elevata dovuta alla natura asincrona del *publish/subscribe* e alla dipendenza da broker esterni. Il server WoT mostra prestazioni temporali superiori, probabilmente grazie a ottimizzazioni specifiche nell'implementazione del protocollo *publish/subscribe*, pur mantenendo overhead nei metadati.

Metrica	Server Tradizionale	Server WoT
Richieste totali	500	500
Tasso di successo	100%	100%
Tempo di risposta medio	14.86 ms	83.30 ms
Tempo di risposta min/max	6-34 ms	29-129 ms
Dimensione pacchetto medio	213.8 bytes	421 bytes
Dimensione pacchetto min/max	212-216 bytes	412-496 bytes

Tabella 5.3: Performance WebSocket: confronto server tradizionale e WoT

Analisi overhead WebSocket

- **Overhead tempo di risposta**: +68 ms
- **Overhead dimensione pacchetto**: +207 bytes

WebSocket mantiene prestazioni eccellenti con overhead contenuto, confermandosi come la scelta ottimale per applicazioni che richiedono comunicazione real-time efficiente.

5.3 Considerazioni sui risultati

Analisi dell'overhead

I risultati evidenziano che l'overhead introdotto dalle specifiche WoT varia significativamente tra i protocolli testati. Il protocollo HTTP presenta un overhead elevato ma accettabile per operazioni non critiche. Il protocollo MQTT mostra un overhead variable, influenzato principalmente da fattori di rete e dalla dipendenza da broker esterni che introduce latenza imprevedibili. WebSocket presenta un overhead moderato con prestazioni più prevedibili.

Benefici vs Costi

L'overhead introdotto dalle specifiche WoT trova piena giustificazione nei vantaggi architetturali che ne derivano. La standardizzazione garantisce interoperabilità universale con qualsiasi client WoT, eliminando la necessità di sviluppare integrazioni custom per ogni tipo di dispositivo. Le Thing Descriptions forniscono metadati completi che permettono l'autodocumentazione dei servizi, riducendo drasticamente i tempi di sviluppo e manutenzione. I meccanismi automatici di scoperta dei dispositivi abilitano la creazione di ecosistemi IoT dinamici e auto-configuranti, mentre il supporto mulit-protocollo trasparente offre flessibilità implementativa senza vincoli tecnologici specifici.

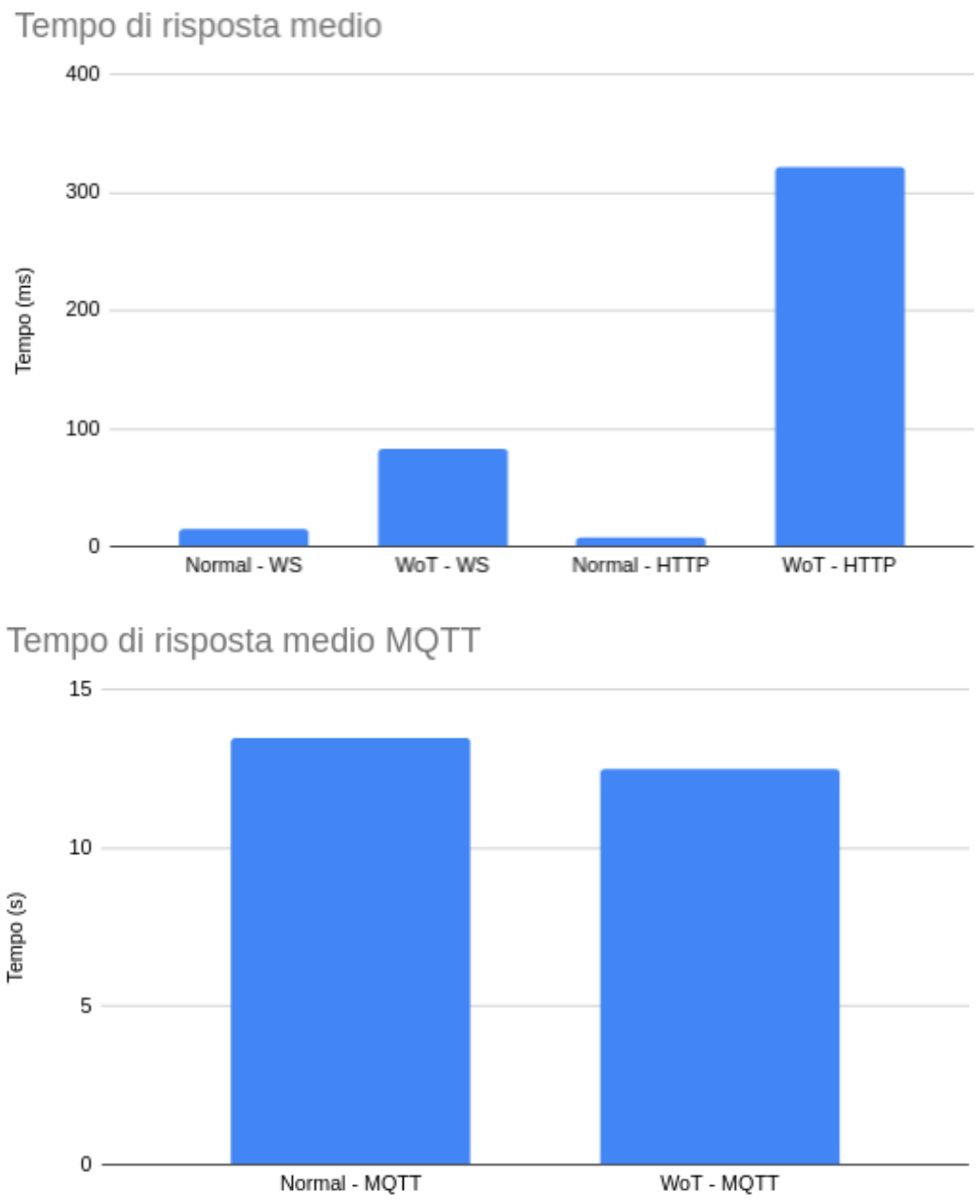

Figura 5.3: Grafici che mostrano comparati i tempi di risposta

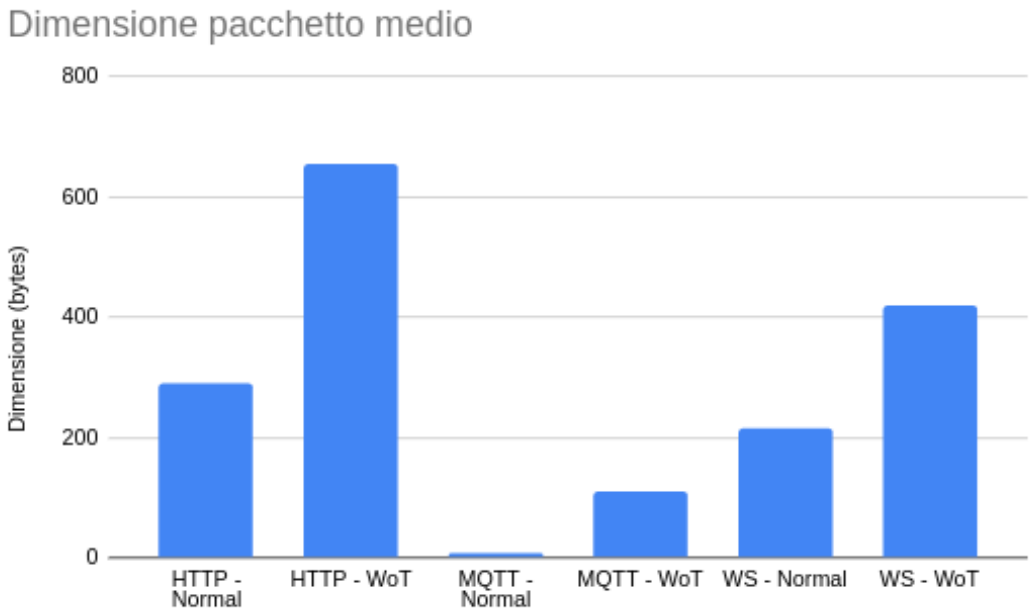

Figura 5.4: Grafico comparativo dimensione dei pacchetti

Raccomandazioni d'uso

L'analisi delle performance orienta verso strategie d'uso diverse per ottimizzare l'efficacia del sistema. WebSocket si configura come la scelta ottimale per applicazioni real-time che richiedono comunicazione a bassa latenza, offrendo prestazioni costanti e prevedibili. HTTP risulta appropriato per operazioni sporadiche o *batch processing* dove l'overhead temporale è accettabile in cambio della semplicità implementativa e compatibilità universale. MQTT trova la sua applicazione ideale in scenari IoT distribuiti caratterizzati da tolleranza alla latenza variabile, dove la robustezza del paradigma *publish/subscribe* compensa le fluttuazioni nelle performance di rete.
I test dimostrano conclusivamente che il Servient WoT Android rappresenta una soluzione tecnicamente valida che bilancia effettivamente standardizzazione e performance, trasformando gli smartphone Android in nodi attivi e pienamente interoperabili nell'ecosistema Web of Things moderno.

Capitolo 6

Conclusione e Sviluppi futuri

Questo lavoro di tesi ha affrontato il problema della frammentazione nell'ecosistema Internet of Things attraverso l'implemeentazione di un Servient WoT completo per dispositivi Android. Partendo dall'analisi dello stato dell'arte delle tecnologie IoT e dell'evoluzione verso il paradigma Web of Things standardizzato dal W3C, è stato sviluppato Android WoT Servient, una soluzione innovativa cha trasforma gli smartphone in nodi attivi dell'ecosistema WoT.
Il progetto ha richiesto uno studio approfondito delle specifiche W3C WoT aggiornate al dicembre 2023 e del framework *kotlin-wot* di Eclipse Thingweb. L'implementazione ha dimostrato con successo la fattiblità tecnica di esporre i sensori Android (accelerometro, giroscopio, magnetometro, sensori di luminosità e pressione) come Interaction Affordances standardizzate, accessibili tramite Thing Description conformi alle specifiche ufficiali. L'architettura sviluppata supporta multipli protocolli di comunicazione (HTTP, MQTT, WebSocket), offrendo flessibilità nelle modalità di accesso e garantendo interoperabilità con qualsiasi client WoT compatibile. L'interfaccia utente integrata permette configurazione dinamica dei sensori, visualizzazione real-time dei dati, invocazione di azioni computazionali e monitoraggio delle performance attraverso grafici interattivi avanzati.
La validazione sperimentale condotta attraverso test comparativi ha fornito una quantificazione precisa dell'overhead introdotto dalle specifiche WoT. I risultati mostrano incrementi nei tempi di risposta variabili dal +537% per WebSocket al +11833% per HTTP, con WebSocket che emerge come il protocollo ottimale per applicazioni real-time. Nonostante l'overhead significativo, l'analisi costi-benefici dimostra che i vantaggi in termini di standardizzazione, interoperabilità e facilità di manutenzione giustificano ampiamente l'investimento computazionale.
Il progetto presenta il primo Servient WoT documentato per Android utilizzando *kotlin-wot*, contribuendo alla democratizzazione dell'accesso alle tecnologie IoT attraverso dispositivi di facile accesso (gli smartphone). Con oltre il 70% del mercato smartphone globale, Android offre un potenziale di scala senza precedenti per l'espansione dell'ecosistema WoT.

6.1 Limitazioni e possibili sviluppi

Limitazioni identificate

Lo sviluppo ha evidenziato diverse sfide tecniche significative. L'overhead computazionale, particolarmente pronunciato per il protocollo HTTP, può limitare l'applicabilità in scenari che richiedono latenza molto bassa. L'implementazione attuale non include meccanismi avanzati di sicurezza, autenticazione o cifratura end-to-end, aspetti critici per deployment produttivi. Inoltre, la gestione energetica non è ottimizzata per un uso prolungato, potenzialmente impattando l'autonomia del dispostivo.
Un limite tecnico significativo è rappresentato da problemi di implementazione nei moduli *kotlin-wot-binding-http* e *kotlin-wot-binding-websocket* che impediscono l'accesso al Servient da dispositivi esterni sulla rete e limitano l'esposizione del servizio esclusivamente su *localhost:8080*. Questi errori di configurazione impediscono la personalizzazione di indirizzi IP e porte, restringendo l'utilizzo del Thing smartphone al solo dispositivo host e compromettendo due dei principali vantaggi dell'approccio WoT: l'accessibilità remota e l'interoperabilità di rete. Entrambi i problemi **sono stati segnalati** ai *maintainer* della repository *kotlin-wot*.

Sviluppi futuri

- **Protocolli**: integrare nuovi protocolli IoT, come ad esempio CoAP.
- **Sicurezza e privacy**: evolvere gli attuali protocolli a versioni più sicure (HTTPS, MQTTS, WSS) e integrare con sistemi di autenticazione, come *OAuth 2.0*.
- **Espansione sensori e azioni**: aggiungere il supporto a più sensori e sviluppo di altre azioni.
- **Discovery**: implementare meccanismi automatici di discovery faciliterebbe la creazione di reti WoT autoconfiguranti.
- **Scripting API**: implementare il building block opzionale WoT Scripting API per aumentare la flessibilità del sistema.
- **User experience e GUI**: migliorare l'esperienza utente attraverso visualizzazioni avanzate, più statistiche e configurazioni guidate.

Bibliografia

[1] IoT Analytics. *State of IoT 2024: Number of connected IoT devices*. https://iot-analytics.com/number-connected-iot-devices/. 2024.

[2] Stefano Pio Zingaro et al. «https://iot-analytics.com/number-connected-iot-devices/». In: *Alma mater studiorum - Università di Bologna - Computer Science and Engineering* (2020).

[3] World Wibe Web Consortium. *Sito W3C*. https://www.w3.org/.

[4] StatCounter - globalstats. *Mobile Operating System Market Share Worldwide*. https://gs.statcounter.com/os-market-share/mobile/worldwide. Accessed: 2024-06-28.

[5] Kevin Ashton. «That 'internet of things' thing». In: *RFID Journal* 7 (2009).

[6] ACS. *Internet of Things: definizione, esempi e applicazioni dell'IoT*. https://www.acs.it/it/blog/digitalizzazione-aziendale/internet-of-things-cos-e/.

[7] MongoDB. *"What is IoT Architecture?"* https://www.mongodb.com/resources/basics/cloud-explained/iot-architecture.

[8] SAP. *"Cos'è l'Internet of Things?"* https://www.sap.com/italy/products/technology-platform/what-is-iot.html?utm_source=chatgpt.com.

[9] IMD. *Smart City Index 2025*. https://www.imd.org/smart-city-observatory/home/.

[10] IETF. *IETF | Internet Engineering Task Force*. https://www.ietf.org/.

[11] Luca Sciullo et al. «A Survey on the Web of Things». In: *IEEE Access* (2022).

[12] Vlad Trifa. *Towards the WOT Manifesto*. https://webofthings.org/2009/04/10/towards-the-wot-manifesto/.

[13] W3C. *"WoT Architecture"*. https://www.w3.org/TR/wot-architecture/.

[14] Dominique Dom Guinard e Vlad M. Trifa. *Building the Web of Things*. Manning, 2016.

[15] Yuchao Zhou et al. «Search Tecniques for the Web of Things: A Taxonomy and Surver». In: *MDPI* (2016).

[16] Eclipse Thingweb. *Eclipse Thingweb*. http://thingweb.io. Accessed: 2024-06-20.

[17] Eclipse Thingwe. *node-wot repository*. https://github.com/eclipse-thingweb/node-wot. Modified: 2025-05-12.

[18] W3C. *"WoT Implementation"*. https://w3c.github.io/wot-marketing/developers/.

[19] Android Studio. *Material Design Hub*. https://m2.material.io/develop/android.

[20] Eclipse Thingweb. *kotlin-wot repository*. https://github.com/eclipse-thingweb/kotlin-wot.

[21] Philipp Jahoda. *MPAndroidChart repository*. https://github.com/PhilJay/MPAndroidChart.