POLITECNICO
MILANO 1863

CAB Product Hub

Applicazione Android per la Gestione di Dispositivi Smart Connessi alla Rete

Relatore:
Luciano Baresi

Collaboratrice:
Manuela Ceroni

Tesi di:
Angelo Falci
Matricola 875123

Anno Accademico 2017-2018

Indice

Sommario

L'obiettivo del progetto seguito in questa tesi è stato quello di realizzare un'applicazione, e una serie di servizi che ruotano intorno ad essa, per monitorare e gestire la raccolta dati sull'utilizzo che gli utenti fanno di alcuni prodotti.
In particolare l'applicazione è stata pensata per interagire con 3 diverse classi di prodotti:

- Prodotti tecnologici dotati di sistemi di connessione diretta con l'applicazione, ad esempio attraverso protocollo bluetooth; a rappresentare questa classe è stato scelto un robot da cucina che si può connettere allo smartphone attraverso protocollo bluetooth
- Prodotti non dotati di alcun tipo di tecnologia, come delle borse o delle scarpe, ai quali vengono applicati vari tipi sensori per la raccogliere dati; a rappresentare questa classe è stata scelta una borsa dentro alla quale è stato posto un beacon, dispositivo dotato di sensori di movimento e che può comunicare dati allo smartphone tramite protocollo bluetooth
- Prodotti tecnologici datati non provvisti di sistema di connessione diretta con l'applicazione sui quali si applicano sistemi di *retrofitting*, cioè si utilizzano tecnologie odierne per renderli oggetti "intelligenti" in grado di connettersi ad altri; a rappresentare questa classe è stata scelta una macchina del caffè non provvista di nessuna tecnologia di connessione

Inoltre l'applicazione è stata pensata sia per raccogliere dati in un ambiente domestico, come in una famiglia, che in contesti più ampi, come all'interno di un ufficio.

Capitolo 1

Introduzione

Basta guardarsi attorno per accorgersi di come gli smartphone siano diffusi al giorno d'oggi, alcune persone ne possiedono anche più di uno. Sono uno strumento indubbiamente utile perché permette di avere a portata di mano innumerevoli servizi sempre con noi.
Ma non sono utili solo a noi consumatori ma anche, e forse ancor di più, a chi i servizi li fornisce. Perché permettono di raggiungere qualsiasi utente in maniera veloce ed economica per farsi pubblicità, per comunicare con lui o monitorare come utilizza un certo prodotto.
Proprio quest'ultimo aspetto sarà al centro del progetto sviluppato in questa tesi, la *raccolta dei dati* al fine di migliorare il prodotto fornito.
A titolo di esempio l'azienda automobilistica Tesla è stata una delle prime grandi imprese a cogliere l'importanza di monitorare le proprie auto raccogliendo dati sull'utilizzo fatto dall'utente al fine di migliorarle e renderle più gradite ai propri clienti.

Prima di andare si vuole chiarire un punto fondamentale, perché non chiedere agli utenti cosa vorrebbero o come migliorerebbero un prodotto anziché raccogliere dati su come lo utilizzano?
La risposta è molto semplice, sarebbero dati raccolti da risposte soggettive e, quindi, contenenti errori.

Il principale problema di monitorare come l'utente utilizzi un certo prodotto è trovare un modo innanzitutto economico, per ovvi motivi, ma anche comodo e rapido per l'utente perché se è necessario, ad esempio, comprare e montare una certa apparecchiatura per permettere il monitoraggio di un prodotto, o causerebbe il rallentamento di una certa attività, ovviamente l'utente (che può anche

essere un intero ufficio) non è motivato a farlo.
La soluzione di base ad entrambi i problemi è lo smartphone, attraverso un'applicazione che, interfacciandosi con il prodotto (ad esempio via bluetooth) raccolga i diversi dati di come l'utente utilizza il prodotto e li invii ad un server dove verranno analizzati. I dati raccolti dipenderanno dai sensori posti nel prodotto stesso e dai permessi che l'utente darà attraverso l'applicazione stessa.

La soluzione sviluppata in questo progetto serve innanzitutto a gestire, utilizzando soltanto uno smartphone, i vari dispositivi per i quali un utente sta raccogliendo i dati e, per ognuno di essi, dare la possibilità all'utente di scegliere quali dati inviare e quali no (ad esempio si può scegliere di inviare i dati riguardante l'utilizzo ma escludere quelli della propria posizione in alcuni casi).

Si può adesso passare ad analizzare meglio il titolo della tesi, CAB Product Hub, titolo preso dal nome del progetto stesso a cui si è partecipati per svolgere questa tesi presso il laboratorio *JOL Milano*, nato dalla collaborazione tra politecnico e TIM.
CAB sta per *Connected product Analytics for Brands* (cioè "Analisi dei prodotti connessi per marchi") che sottolinea l'obiettivo che vuole raggiungere questo progetto, la raccolta di dati su diverse tipologie di prodotti, che quindi non si limitano ad un unico brand.
La seconda parte del titolo, *Product Hub*, che potremmo tradurre come "hub per prodotti", sottolinea la caratteristica chiave dell'applicazione sviluppata nel corso della tesi. Cioè avere una soluzione che, come un hub fisico, abbia più porte e possa collegarsi e ricevere dati da più prodotti, dati che saranno poi inoltrati ad un server che li raccoglierà in un database.
Per concludere potremmo tradurre l'intero titolo come "soluzione hub che inoltri i dati di prodotti provenienti da più marchi in un unico server".
Si utilizza il termine "soluzione" perché non ci si riferisce solamente all'applicazione android ma anche a tutti i servizi che la circondano e, in alcuni caso, si connettono a prodotti che lo smartphone direttamente non può raggiungere.

L'iterazione che avrà poi l'applicazione con i vari dispositivi che verranno aggiunti in essa dipenderà dalla tecnologia posseduta da essi. In particolare in questo progetto ci si è occupati di gestire tre diversi classi di dispositivi e, per ognuna di queste classi, è stato scelto uno specifico prodotto che la

rappresentasse:

- Dispositivi tecnologici dotati di sistemi di connessione diretta con l'applicazione, ad esempio attraverso protocollo bluetooth; a rappresentare questa classe è stato scelto un robot da cucina che si può connettere allo smartphone attraverso protocollo bluetooth

- Dispositivi non dotati di alcun tipo di tecnologia, come delle borse o delle scarpe, ai quali vengono applicati vari tipi sensori per la raccogliere dati; a rappresentare questa classe è stata scelta una borsa dentro alla quale è stato posto un beacon, dispositivo dotato di sensori di movimento e che può comunicare dati allo smartphone tramite protocollo bluetooth

- Dispositivi tecnologici datati non provvisti di sistema di connessione diretta con l'applicazione sui quali si applicano sistemi di *retrofitting*, cioè si utilizzano tecnologie odierne per renderli oggetti "intelligenti" in grado di connettersi ad altri; a rappresentare questa classe è stata scelta una macchina del caffè non provvista di nessuna tecnologia di connessione

Per quest'ultimo dispositivo è stata utilizzata una presa intelligente che manda ad un server OpenHAB i dati della corrente istantanea assorbita dalla macchina ogni tre secondi. Sul server un algoritmo creato ad-hoc, scritto dopo l'analisi dei valori di potenza letti, è in grado di riconoscere lo stato della macchina del caffè riuscendo ad esempio a distinguere quando viene fatto un caffè. Gli eventi generati vengono quindi inoltrati al server che raccoglie tutti i dati

Si vuole sottolineare un ultimo aspetto prima di presentare l'organizzazione della tesi. I dati raccolti vengono associati, nel database, all'utente stesso che li raccoglie. Questo è importante per l'analisi successiva dei dati perché si possono fare importanti valutazioni altrimenti impossibili, come ad esempio calcolare la media di quante volte un utente faccia un caffè.
Dato che si raccolgiono dati riguardo prodotti che spesso vengono usati da più persone (ad esempio il robot da cucina che può essere usato dai membri di una famiglia) ciascun account creato sull'applicazione potrà possedere più profili, potrà quindi essere associato a più persone. Solo uno di questi profili per volta sarà attivo, che potrà essere cambiato quando si vuole, e ad esso verranno associati i dati raccolti. Maggiori dettagli verranno dati nel capitolo dedicato all'applicazione (capitolo 5).

Dopo questa introduzione come prima cosa si vedrà una generale panoramica delle tematiche affrontate (capitolo 2) da questo progetto per poi passare ad una generale descrizione delle tecnologie e dispositivi utilizzati (capitolo 3).
Successivamente verrà presentato e descritto l'intero progetto, dedicando un capitolo al sistema messo in piedi per la macchina del caffè (capitolo 4) e uno, il più corposo della tesi, dedicato all'applicazione Android sviluppata (capitolo 5).
Infine verranno tratte delle conclusioni (capitolo 6) sulla soluzione finale sviluppata e su casi reali di utilizzo nella quale la si potrà vedere utilizzata.

Capitolo 2

Tematiche Affrontate e Stato dell'Arte

2.1 Internet delle Cose

In telecomunicazioni Internet delle cose (o, più propriamente, Internet degli oggetti o IoT, acronimo dell'inglese Internet of Things) è un neologismo riferito all'estensione di Internet al mondo degli oggetti e dei luoghi concreti.
Si tratta di un'evoluzione dell'uso della rete in cui gli oggetti (le "cose") acquisiscono "intelligenza" connettendosi alla rete, rendendosi così riconoscibili e potendo comunicare con altri dispositivi. In questo modo si può avere uno scambio continuo di informazioni tra i diversi oggetti connessi migliorando l'efficienza di ognuno di essi. I campi di applicabilità sono innumerevoli, dalle applicazioni industriali (processi produttivi), alla logistica, fino all'efficienza energetica, all'assistenza remota e alla tutela ambientale.[8]
Alcuni esempi banali in cui l'IoT potrebbe migliorare le attuali tecnologie più diffuse:

- La sveglia suona prima in caso di traffico
- La batteria elettrica dell'auto, che si ricarica ad energia solare, viene usata parzialmente la sera per consumi domestici se nell'agenda dell'utente non vi sono segnati impegni per il mattino successivo
- Un rilevatore del battito cardiaco fa partire una chiamata d'emergenza con la propria posizione se rileva un'anomalia nel battito

- Una lavatrice che rileva un malfunzionamento nei propri componenti e, in automatico, prenota un appuntamento con un assistente nei giorni in cui il proprietario è in casa (quest'ultima informazione ricevuta dall'agenda connessa ad internet)

2.1.1 IoT in Italia

Nel 2017 il mercato del IoT ha toccato i 3,7 miliardi di euro, con una crescita del 32% rispetto al 2016, in linea con l'anno precedente. Crescita che conferma la centralità dell'IoT nello sviluppo digitale del nostro paese.
I contatori del gas intelligenti, i cosiddetti *smart metering*, sono al primo posto tra gli oggetti "intelligenti" introdotti in Italia (tra l'altro unico paese in Europa ad introdurli in maniera massiva[9]). Al secondo posto abbiamo le *smart car* che registrano un forte incremento sia di auto connesse con box GPS/GPRS sia di auto connesse nativamente. Negli ambiti applicativi abbiamo una crescita considerevole dei progetti di *smart building* sopratutto per soluzioni legate alla sicurezza degli edifici.
Trovano invece ancora poco sbocco le applicazioni *smart city e smart environment* fatta eccezione per alcune applicazioni ben circoscritte, ad esempio il trasporto pubblico (con mezzi monitorati da remoto) e per l'illuminazione intelligente.
Ultimo interessante dato è costituito dalla diffusione fisica dell'IoT che a fine 2016 contava 14,1 milioni di oggetti connessi in Italia tramite rete cellulare (+37% rispetto all'anno precedente).[10]

2.2 IoT e Big Data

La raccolta dei dati è l'obiettivo principale del progetto sviluppato in questa tesi.
In diversi settori si raccolgono e analizzano dati per le più disparate ragioni, per migliorare un algoritmo di ricerca o per ottimizzare un processo industriale. Comunque, come già menzionato nell'introduzione, in questo progetto ci si concentrerà sulla raccolta di dati riguardanti l'utilizzo che fanno gli utenti di determinati prodotti.

2.2.1 Come raccogliere i dati

Come si inizia a raccogliere i dati? In questa tesi non ci occuperemo di analizzare le componenti hardware che permettono la raccolta dei dati, è tuttavia utile spendere due parole su tale argomento per dare un quadro più generale possibile dell'intero problema affrontato.
Durante la raccolta dei dati la tentazione è quella di riempire il sistema che si vuole analizzare (per una qualsiasi ragione) con una miriade di sensori e sistemi di acquisizione che inviino fiumi di dati ad un sistema di raccolta. In realtà questo è un approccio completamente sbagliato perché, oltre ad aumentare il costo delle spese, rischia di riempire il database di dati inutili inquinando quelli veramente rilevanti e rallentando l'efficienza dell'algoritmo che dovrà analizzarli. Il corretto approccio consiste innanzitutto nel selezionare, tra tutti i dati generati da un certo dispositivo, solo quelli veramente rilevanti in base a cosa si vuole ottenere, creando un modello matematico/informatico del processo (ad esempio se si vogliono utilizzare dei beacon ai soli fini di localizzazione è inutile usare beacon che inviino dati riguardo i loro movimenti). E in base a questa analisi si sceglieranno i dispositivi hardware adeguati alle proprie esigenza e si raccoglieranno da essi i dati considerati utili.[11]

2.2.2 Quali tecnologie si usano in Italia

I nuovi big data, caratterizzati da grosse moli di dati complessi ed eterogenei tra loro, rendono obsolete le vecchie tecnologie di data warehousing e impongono nuovi approcci architetturali. In sintesi questo significa:

- Disporre di un'infrastruttura scalabile in grado di elaborare grosse moli di dati in tempo reale
- Appoggiarsi ad architetture in gradi di unire elaborazione in tempo reale e in batch nella fase di analisi
- Per la fase di conservazione dei dati implementare soluzioni in grado di gestire le più svariate fonti informative e garantirne l'immediata disponibilità al bisogno

L'integrazione fra i dati raccolti da sistemi diversi è un elemento cardine di questo disegno.
Ma sono necessarie anche soluzioni di data management studiate ad hoc per gestire vari dati con processi sia relazionali che non, utilizzando machine learning o linguaggi di programmazione come *Python* o *R*.

Relazione tra open source e big data

Secondo *L'Osservatorio Big Data Analytics e Business Intelligence del Politecnico di Milano* l'open source "è considerato uno dei fattori abilitanti la diffusione dei big data". Infatti il 30% (in crescita) delle aziende coinvolte nella ricerca utilizza software provenienti dal mondo open source, ed è più significativo di quello che sembra contando che tante soluzioni private utilizzano componenti open source al loro interno.

Organizzazione dei dati

Prima della diffusione dei big data i dati venivano normalmente immagazzinati e gestiti da database di tipo relazionale. La memorizzazione attraverso schemi tabellari ha però delle rigidità che rendono difficile il suo utilizzo nel caso di big data. Per risolvere questo problema è nato un movimento di *NoSQL* (NotOnlySQL) che utilizza sistemi che consentono di scalare orizzontalmente (e non più solo verticalmente).
Recentemente si sono sviluppate nuove classi di database, chiamati NewSQL, che nonostante implementino anch'essi modelli relazionali, come i classici SQL, garantiscono performance paragonabili a quelle offerte dai sistemi NoSQL.

Integrazione dei dati

Le aziende in passato si sono creati numerosi silos per raccogliere insiemi di dati aziendali. Con i big data però si è reso necessario poter archiviare i dati nel loro formato nativo e si è per questo diffuso il concetto di data lake. Nelle aziende italiane si utilizzano oggi quattro modalità di archiviazione dei dati:

- **Silos:** approccio tradizionale dove lo spazio è suddiviso in repository in base ai dipartimenti aziendali con ambienti isolati tra loro
- **Data Warehouse:** archivio informatico che raccoglie i propri dati aziendali integrandoli con dati provenienti dall'esterno (database per dati strutturati)
- **Data Lake:** i dati vengono archiviati nel loro formato nativo, dato che non è necessario dar loro una struttura, rendendo così possibile l'integrazione di elevate quantità di dati di qualsiasi formato e provenienti da più fonti

- **Modello Integrato:** organizzazione in cui si ha a disposizione sia un data lake che un data warehouse che lavorano in modalità integrata per soddisfare le differenti esigenze di storage, gestione e analisi

Modello di sourcing adottato

Per le attività di analisi le aziende si avvalgono di supporti esterni, tra questi si identificano tre principali modalità di sourcing:

- **outsourced:** l'azienda semplicemente si rivolge all'esterno cercando un fornitore che offra servizi per l'analisi dei dati
- **on premises:** l'azienda crea una propria infrastruttura
- **cloud:** si sfrutta l'erogazione, in modalità *IaaS o PaaS*, dell'infrastruttura tecnologica (tutta o una parte di essa) per fruire della potenza di calcolo di cui si ha bisogno

[12]

2.3 Un'App per Connettersi a Tutto

Con la sviluppata in questo progetto si vuole soddisfare l'esigenza di dover raccogliere dati riguardo molteplici prodotti e, quindi, usando diverse tecnologie di raccolta e trasferimento dati. Inoltre con l'applicazione si potranno gestire un insieme di prodotti in un unico punto.
All'interno dell'applicazione sono state inserite delle librerie, sviluppate separatamente e poi integrate nell'applicazione. Utilizzando tali librerie, già ampiamente testate durante questo progetto, si potranno realizzare rapidamente eventuali altre applicazioni, dedicate magari ad uno specifico produttore.

Capitolo 3

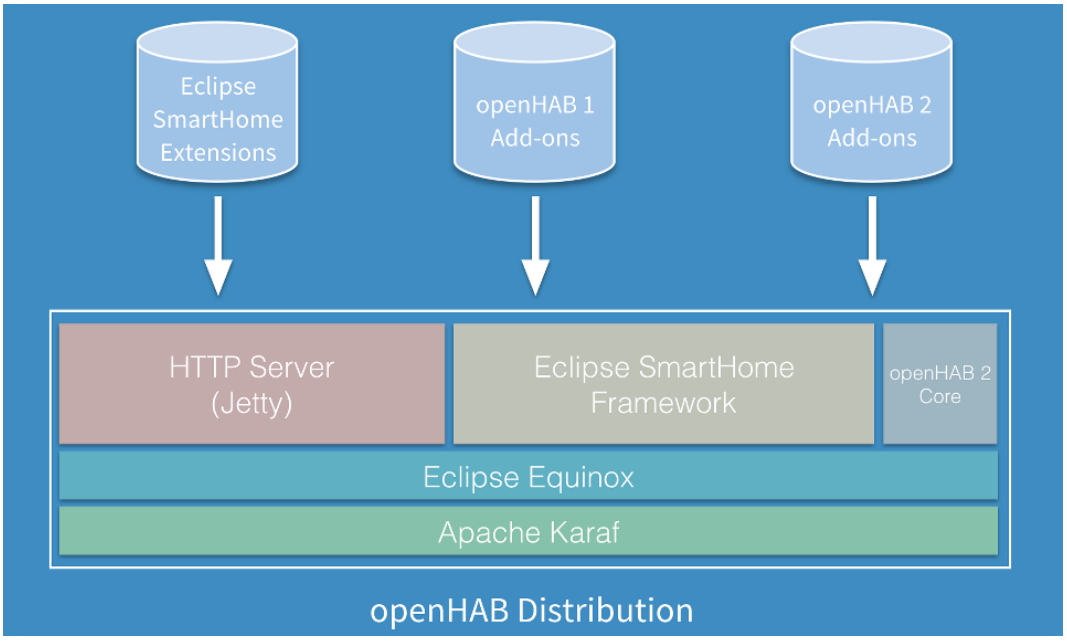
Tecnologie Utilizzate

In questo capitolo verranno presentati le principali tecnologie e dispositivi utilizzati in questo progetto e, per ognuno di essi, una breve descrizione, i motivi per cui è stata fatta questa scelta, se necessari, e i vantaggi che comporta.

3.1 OpenHAB

OpenHAB ("open Home Automation Bus") è una piattaforma di domotica open source dalla quale si possono controllare vari dispositivi presenti in casa. Questo grazie al software di OpenHAB che è in grado di integrare diversi sistemi di automazione domestica, dispositivi e tecnologie in un'unica soluzione.[7]
Si è scelto OpenHAB per gestire il *sistema di retrofitting* in quanto rappresenta lo stato dell'arte per quanto riguarda la domotica, unendo semplicità e grande efficienza.
È sviluppato in Java, principalmente basato sul framework *Eclipse SmartHome*, e può quindi girare su qualsiasi sistema operativo. Inoltre è un software altamente modulare il che permette di essere esteso facilmente attraverso degli *Add-ons* per interagire con un infinità di diversi *oggetti intelligenti*.

Figura 3.1: Architettura openHAB



In particolare si è usato OpenHAB per raccogliere i dati dalla presa intelligente collegata alla macchina del caffè.

3.2 Server CentOS

Nel server che contiene il database che raccoglie e salva tutti gli eventi abbiamo utilizzato il sistema operativo CentOS. Si è scelto questo sistema operativo, preferendolo alla versione server di Ubuntu ad esempio, perché è da sempre sviluppato con una filosofia tendenzialmente conservatrice, preferendo la stabilità e sicurezza all'innovazione (che porta sempre con sé qualche rischio).[3] Uno dei maggiori vantaggi di CentOS è che di default qualsiasi richiesta proveniente dall'esterno viene rifiutata rendendo la gestione della sicurezza automatica.

3.3 MongoDB

MongoDB è un DBMS non relazionale, orientato ai documenti. Classificato come un database di tipo NoSQL, MongoDB si allontana dalla struttura tradizionale basata su tabelle dei database relazionali in favore di documenti in stile JSON con schema dinamico rendendo l'integrazione di dati, in alcuni tipi di applicazioni, più facili e veloci.[18]
Si è scelto di utilizzare MongoDB, in alternativa ad altri DBMS, sia perché non è un database relazionale sia per la sua flessibilità a integrare diversi tipi di dati. In particolare lo si è usato nel server principale per salvare al suo interno gli eventi generati dagli utenti mentre interagiscono con i vari prodotti connessi alla soluzione.// MongoDB è l'ideale per questo progetto per:

1. La sua struttura, che favorisce i documenti in stile JSON, perché dall'applicazione gli eventi vengono inviati al server proprio tramite delle *authetication POST* che hanno come body proprio un JSON

2. La sua flessibilità a interagire con diverse tipologie di dati perché oggetti diversi inviano eventi contenenti dati diversi tra loro, impedendo quindi l'utilizzo di un approccio relazionale dove le tabelle, aventi una rigida struttura, possiedono colonne in grado di salvare dati della stessa tipologia

3.4 NGINX

NGINX (pronunciato come "engine-x") è un web server/reverse proxy leggero ad alte prestazioni. Fornisce rapidamente i contenuti statici con un utilizzo efficiente delle risorse di sistema.[4]
È stato installato e configurato NGINX come web server per l'applicativo web creato per la macchina del caffè (nel capitolo 4 i dettagli). Si è optato per questo web server perché ritenuto il più semplice e veloce per la soluzione che si voleva ottenere.

3.5 Java

In informatica Java è un linguaggio di programmazione ad alto livello, orientato agli oggetti e a tipizzazione statica, specificatamente progettato per essere il più possibile indipendente dalla piattaforma di esecuzione. Purtroppo tale caratteristica comporta perdita di prestazioni rispetto a linguaggi compilati

come c++.[19]
Si è utilizzato Java per programmare su OpenHAB (3.1) in quanto quest'ultimo è implementato in questo stesso linguaggio. È stato utilizzato in particolare per realizzare un *binding* ad-hoc per connettere la presa intelligente, connessa alla macchina del caffè, con il server in cui è installato OpenHAB.

3.6 Android Studio

Per sviluppare l'applicazione descritta in questo documento si è deciso di utilizzare Android Studio come ambiente di sviluppo. Si è optato per Android Studio, in alternativa ad altri editor, perché essendo lo strumento ufficiale per lo sviluppo di applicazioni Android gode di innumerevoli vantaggi:

- *Ampio supporto* sia perché la stessa azienda Google si occupa di supportarlo e aggiornarlo, sia perché utilizza di base linguaggi come Java, Kotlin e XML (per la grafica) usati da molti anni e ampiamente diffusi nel panorama. Questo vuol dire poter trovare innumerevoli guide, sia ufficiali che non, realizzate ad hoc per specifici compiti (come ad esempio la libreria sui beacon)

- *Non ha limitazioni* al contrario di tante altre soluzioni presenti sul mercato, spesso molto più semplici e con un interfaccia intuitiva da usare, soprattutto per chi si avvicina alla programmazione android, ma con grosse limitazioni

- *Larga compatibilità* proprio perché Android Studio è nato esclusivamente per applicazioni su dispositivi Android garantisce una maggiore compatibilità su differenti dispositivi e versioni del sistema operativo stesso (compatibilità che ovviamente dipende anche dalle impostazioni settate dal programmatore)

3.7 Kotlin

Kotlin è un linguaggio di programmazione general purpose, multi-paradigma, open source sviluppato dall'azienda di software JetBrains. Si basa sulla JVM (Java Virtual Machine) ed è ispirato ad altri linguaggi di programmazione tra i quali Scala e lo stesso Java, mentre ulteriori spunti sintattici sono stati presi da linguaggi classici, come il Pascal e moderni come Go.[1]
Si è scelto di utilizzare Kotlin, in alternativa a Java, perché, oltre ad essere completamente compatibile con Java, ha rispetto a quest'ultimo diverse migliorie grazie al fatto di essere un linguaggio nato molto

più di recente e quindi senza tutti i piccoli errori commessi in passato su Java.
I principali vantaggi sono:

- Ha la capacità di evitare alcune classi di errore, come gli accessi a puntatori nulli
- Ha una sintassi che permette di scrivere uguali funzioni utilizzando meno righe di codice rispetto a Java lasciando il codice più pulito e leggibile
- La piena compatibilità con Java, le migliorie rispetto a quest'ultimo e il fatto che Google abbia dichiarato Kotlin come linguaggio ufficiale Android al pari di Java fa supporre che sia il linguaggio del futuro e che possa addirittura sostituire Java per quanto riguarda lo sviluppo mobile ed è quindi un chiaro vantaggio usarlo per un progetto che, se avrà successo, potrà avere futuri sviluppi

3.8 Python e Jupyter

Python è un linguaggio multi-paradigma orientato agli oggetti che ha fra i principali obiettivi dinamicità, semplicità e flessibilità.[13]
Si è scelto di utilizzare tale linguaggio, in combinazione con Jupyter, per elaborare e analizzare i dati provenienti dalla macchina del caffè, sia quelli ricavati dall'applicativo web sia quelli ricavati tramite il monitoraggio della potenza assorbita. Si è scelto Pyhton perché dispone di librerie, *pandas* prima tra tutte in questo caso, che permettono una manipolazione dei dati contenuti in file csv veramente molto rapida tramite l'utilizzo di poche righe di codice.

3.9 HTML, CSS e JavaScript

Si sono utilizzati HTML, CSS e JavaScript, cioè i linguaggi base per implementare una qualsiasi pagina web statica, per sviluppare l'applicativo web che raccoglie i dati dei caffè.

3.10 Protocollo Bluetooth

Bluetooth (spesso abbreviato in BT) è uno standard tecnico-industriale di trasmissione dati per reti personali senza fili (WPAN: Wireless Personal Area Network). Fornisce un metodo standard, economico e sicuro per scambiare informazioni tra dispositivi diversi attraverso una frequenza radio sicura

e a corto raggio in grado di ricercare i dispositivi coperti dal segnale radio entro un raggio di qualche decina di metri mettendoli in comunicazione tra loro. Questi dispositivi possono essere dei più svariati, ad esempio palmari, telefoni cellulari, personal computer, portatili, stampanti, fotocamere digitali, smartwatch, console per videogiochi purché provvisti delle specifiche hardware e software richieste dallo standard stesso.
Si è utilizzato questo tipo di connessione per far comunicare l'applicazione con il robot da cucina e per leggere lo stato del beacon.

3.11 Tecnologia NFC

La tecnologia *Near-Field Communication* (traducibile con "comunicazione di prossimità") fornisce connettività senza fili, bidirezionale a corto raggio (fino ad un massimo di 10cm).[6]
Tale tecnologia trova largo uso in operazioni di pagamento o riconoscimento di una persona attraverso, ad esempio, lo scorrimento di un badge (dotato di NFC) in un dispositivo di lettura NFC.
Si è utilizzata questa tecnologia per sfruttare la lettura di badge da parte dell'applicazione per cambiare il profilo attivo al quale associare i dati raccolti.

3.12 Beacon

I beacon, "fari" in inglese, sono piccoli dispositivi che funzionano con tecnologia BLE (Bluetooth Low Energy) e sono uno dei dispositivo hardware più usati per fare marketing di prossimità. Le tecnologie di localizzazione e di comunicazione attraverso dispositivi mobile stanno avendo oggi un grande successo, grazie soprattutto all'importanza che i mobile device, quali smartphone e tablet, hanno acquisito nella vita delle persone.[2]
In base ai sensori messi all'interno di un beacon si possono rilevare diverse informazioni. I beacon che si sono utilizzati in questo progetto sono rilevabili dall'applicazione, tramite bluetooth, che ne legge lo stato per capire se si stanno muovendo o sono fermi. In questo modo si può in generale monitorare per quanto e in quali periodi della giornata un utente usa un certo prodotto a cui il beacon e attaccato

fisicamente (come ad esempio un capo d'abbigliamento o una borsa).

3.13 Robot da Cucina

Si tratta di un robot da cucina multifunzione che può essere programmato attraverso dei pulsanti e un display posti nella parte frontale. Cucina pietanze che richiedono lunga cottura, cuoce al vapore, mescola, salta, frulla, impasta e sbatte in completa autonomia.
Inoltre è possibile collegarsi alla pentola via bluetooth, con un solo smartphone per volta, e ricevere dati da essa.
Per la pentola esiste anche un'applicazione ufficiale (non citabile per motivi di privacy) che contiene una lista di ricette realizzabili con la pentola e inoltre, collegandosi ad essa, mostra che programma sta eseguendo la pentola.
Nel progetto l'applicazione si collega direttamente alla pentola dalla quale riceve gli eventi che l'utente ha attivato premendo i pulsanti.

3.14 Macchina del Caffè e Presa Intelligente

Una comune macchina del caffè che funziona a cialde con la quale è possibile fare un caffè corto o lungo. Non è dotata di nessuna tecnologia di connessione, per tale motivo è collegata alla presa elettrica attraverso una presa intelligente in grado di misurare ed inviare ad un server, ad ogni intervallo di tempo, i livelli di potenza della corrente che la attraversa. Tali livelli ovviamente dipenderanno dall'uso che si sta facendo della macchina durante la lettura.
È l'unico dispositivo non connesso direttamente all'applicazione, dalla quale è possibile soltanto monitorare i permessi riguardo le informazioni mandate dalla macchina al server.

Capitolo 4

Sistema di Retrofitting

openHAB e Presa Intelligente

La macchina del caffè non è dotata di alcuna tecnologia per la connessione e per tale motivo non è possibile collegarla direttamente all'applicazione, ad esempio attraverso bluetooth. Per riuscire a monitorare il suo utilizzo la si è collegata ad una presa intelligente che monitora la potenza assorbita dalla macchina e invia i livelli letti ogni 3 secondi (intervallo di tempo che può essere variato) ad un server, in cui è installato OpenHAB.
Si noti bene che il server su cui è installato OpenHAB può coincidere con il server che raccoglie i dati. In caso siano due macchine diverse il server in cui è installato OpenHAB manderà i dati raccolti all'altro.
Inizialmente si è scritta una regola su OpenHAB, tramite un linguaggio simile ad un JavaScript semplificato, che identificasse lo stato istantaneo della macchina del caffè in base alla potenza istantanea assorbita da essa e, in alcuni casi, allo stato precedente.
Tuttavia la potenza assorbita dalla macchina non è costante e i picchi che si raggiungono quando la si utilizza non sono uguali tra loro. Inoltre quando si accende la macchina inizialmente si ha un picco solitamente più alto degli altri. Tutto ciò ha come conseguenza che contare il numero di caffè bevuti basandosi sulla potenza assorbita istantanea dà grossi errori.
Un ultimo problema è che, al contrario del beacon e del robot da cucina, per la macchina del caffè non si riescono a collegare gli eventi inviati con le persone che le hanno generate.
Innanzitutto per risolvere il problema della potenza non costante si è realizzato un *binding* ad-hoc in Java che, grazie ad un linguaggio più completo, permettesse di creare un algoritmo più complesso in

grado di monitorare lo stato della macchina del caffè con maggior precisione.
Per arrivare a tale risultato tuttavia si è affrontato il problema un passo per volta:

1. Identificando gli stati che si volevano distinguere
2. Creando un applicativo web per raccogliere dati su quanta potenza assorbisse la macchina nei vari stati in cui si poteva trovare
3. Analizzando i dati raccolti
4. Creando un algoritmo per distinguere i diversi stati

Per risolvere il secondo problema, associare cioè i dati raccolti alle persone che li hanno generati, si è utilizzata l'applicazione stessa sfruttando, in alcuni casi, anche la tecnologia NFC (maggiori dettagli verranno dati nella sezione dedicata 4.4.1).

4.1 Identificazione dell'obiettivo

Innanzitutto, per monitorare il comportamento della macchina, si è deciso di trattare il problema attraverso una macchina a stati. Quindi "monitorare" prende il significato di *trovare in che stato si trova la macchina* e si sono identificati i seguenti stati:

- **off** quando la macchina è spenta
- **heating** quando la macchina sta scaldando l'acqua e avviene sempre dopo essere stata accesa o quando finisce del tutto l'acqua nel serbatoio interno e arriva quella nuova che è tutta fredda
- **ready** dopo che l'acqua è stata scaldata la macchina è pronta per fare il caffè
- **short coffee** mentre la macchina sta facendo un caffè corto
- **long coffee** mentre la macchina sta facendo un caffè lungo

Nei test fatti dopo non si è però tenuto conto di quest'ultimo stato.

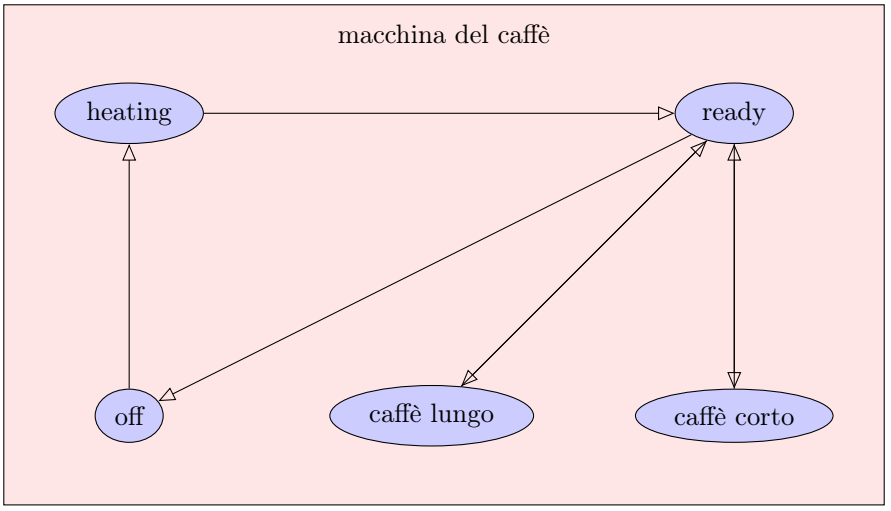
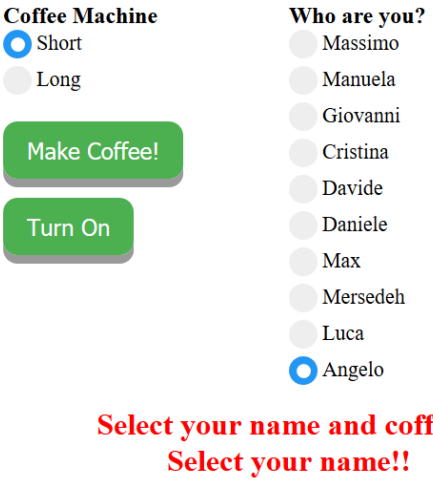


Figura 4.1: Stati della macchina del caffè

4.2 Applicativo Web

Una volta identificato il problema bisognava raccogliere dei dati per capire qual'era la potenza media assorbita dalla macchina durante gli stati sopra menzionati.

Per fare questo si è realizzato un applicativo web utilizzato dalle persone dell'ufficio, i quali premevano i rispettivi pulsanti ogni volta che accendevano la macchina o si facevano un caffè.



In questo modo è stato possibile raccogliere una serie di dati sul server che indicavano gli orari in cui la macchina stava facendo un caffè o era stata appena accesa.

4.3 Analisi dei dati

Successivamente si è fatta un'analisi dei dati confrontando quelli raccolti con l'applicativo web con quelli della potenza per studiare il comportamento della macchina e, in particolare, quanta potenza assorbiva mentre faceva un caffè o veniva accesa.
Per farlo si è utilizzato il linguaggio *Python* sul web application *Jupyter* e si sono anche rappresentati i dati graficamente. Da quest'ultimi si è osservato che i picchi variano in maniera troppo brusca e quindi basandosi su di essi non si può avere la precisione desiderata.
Si è deciso così di considerare un intervallo di valori, cioè di picchi, anziché uno singolo, per determinare lo stato della macchina. Di questo intervallo si è calcolata la media della potenza. Sono stati considerati gli intervalli intorno agli eventi *heating* e *caffè corto*.
Per conoscere invece la potenza assorbita mentre la macchietta non veniva usata, ed era quindi in stato di *ready*, sono stati considerati intervalli lontani dagli eventi citati precedentemente.
Come lunghezza dell'intervallo considerato si è provato con 15 secondi (che è circa il tempo impiegato dalla macchina per fare un caffè corto) e durante i test fatti si è rivelato un buon valore. Successivamente si è ipotizzato di aumentare tale intervallo di tempo a 20 secondi (circa il tempo necessario per un caffè lungo) in modo da riuscire a distinguere le due tipologie di caffè. In questo modo sarebbe stato facile distinguere il caffè lungo perché, dato che dura di più dei 15 secondi di quello corto, aveva una potenza media sull'intervallo maggiore. Tuttavia tale ipotesi, per quanto validata dai dati, non è stata testata.
Nei seguenti grafici nell'asse delle X abbiamo il tempo mentre nell'asse delle Y possiamo vedere in:

- **Rosso** i livelli di potenza assorbiti dalla macchina del caffè in mA
- **Arancione** gli istanti in cui qualcuno ha premuto nell'applicativo web il pulsante per aver acceso la macchina (il valore 1000 è fittizio e serve solo per sovrapporlo ai valori numerici della potenza)
- **Azzurro** gli istanti in cui qualcuno ha premuto nell'applicativo web il pulsante per aver fatto un caffè con la macchina (anche in questo caso il valore 1000 è fittizio e serve solo per sovrapporlo ai valori numerici della potenza)

Come si può anche vedere dai valori indicati sull'asse delle X il grafico più in basso è una porzione ingrandita del primo grafico.

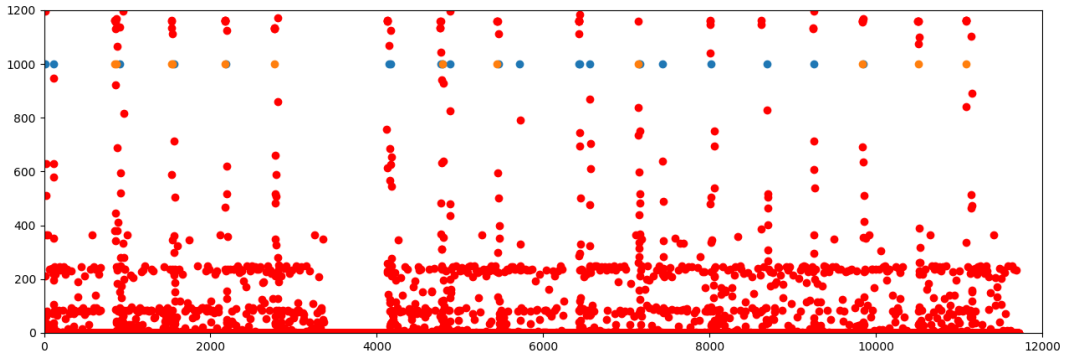


Figura 4.2: Grafico 1: dati della macchina del caffè

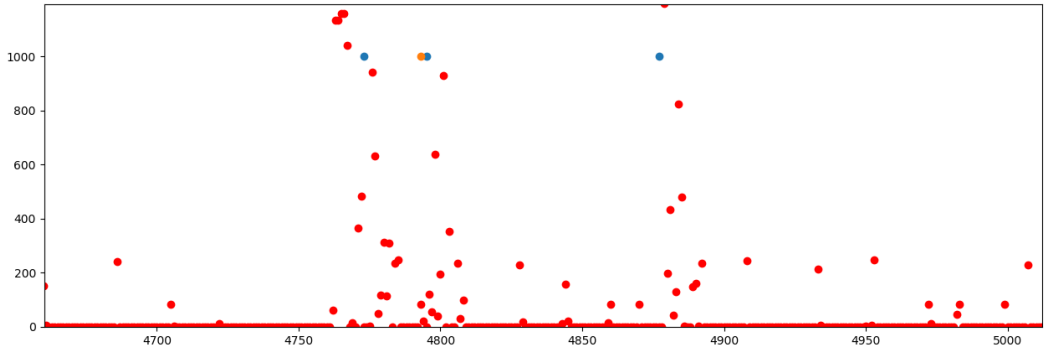


Figura 4.3: Grafico 2: dati della macchina del caffè

Dai dati analizzati si sono ricavati i seguenti valori medi della potenza:

- Da 1000 a 1200 mA di potenza assorbita la macchina è nello stato di *heating* (sta scaldando l'acqua)
- Da 200 a 300 mA di potenza assorbita la macchina è nello stato di *caffè corto*
- Da 1 a 50 mA di potenza assorbita la macchina è nello stato di *ready* (l'acqua è stata scaldata e la macchina è pronta per essere usata)

- 0 quando la macchina è spenta; tale valore non viene di fatto ricevuto perché se la macchina non assorbe più corrente anche la presa intelligente risulta spenta

4.4 Algoritmo di monitoraggio

Tale algoritmo è stato scritto sul binding dedicato alla macchina del caffè per riconoscere i vari stati in cui essa può trovarsi.

Cos'è un binding in openHAB?

In parole semplici si tratta di un componente software, sviluppato in Java, che mette in comunicazione un componente hardware con il resto del codice, permettendo di ricevere dati o comandi dal componente hardware e, a sua volta, inviarne. Nel caso della macchina da caffè, o meglio la presa intelligente ad essa collegata, il binding può solo ricevere i dati riguardo la potenza in ingresso e non può inviare nulla dato che il componente hardware non è in grado di elaborare dati in ingresso.

Tramite l'analisi dei dati raccolti con l'applicativo web si è costruito un algoritmo ad-hoc per monitorare l'utilizzo che un utente fa della macchina distinguendo i diversi stati menzionati nella sezione precedente.

Come funziona l'algoritmo

Per ogni stato che si vuole identificare è stato istanziato un vettore (tranne che per lo stato di *off*). La lunghezza del vettore determina su che periodo stiamo calcolando la media. Ad esempio, sapendo che viene inviato un campione ogni 3 secondi dalla presa intelligente, avere un vettore lungo 5 vuol dire considerare un intervallo di 15 secondi.
Ciascun stato ha un vettore personale perché per distinguere 2 diversi stati non sempre è necessario calcolare la media sullo stesso numero di campioni. Ad esempio per un caffè è indispensabile calcolare la media su un intervallo di 15 secondi, che è il tempo impiegato dalla macchina per fare un caffè, mentre per valutare se siamo nello stato di *ready* potrebbero bastare solamente 3 campioni dato che l'assorbimento della macchina in questo stato è più basso e costante.

Lo stato di partenza della macchina del caffè all'avvio dell'algoritmo è di default *off*. Stanziati poi i vettori l'algoritmo, ogni volta che arriva un nuovo valore di potenza letto:

1. Inserisce il nuovo valore in tutti i vettori istanziati; se un vettore è pieno elimina il valore più vecchio raccolto e lo sostituisce con quello nuovo
2. Per ogni vettore pieno, associato ad uno stato che non sia l'attuale, viene calcolata la media
3. Se il valore della media di un certo vettore rientra nei range che dovrebbe avere lo stato associato a quel vettore allora la macchina cambia stato e viene inviato un nuovo evento al server con il cambio di stato
4. Se la macchina ha cambiato stato vengono svuotati tutti gli array prima di rimettersi in attesa di un nuovo valore; questo accorgimento evita ad esempio che, dopo lo stato *heating*, ricevendo due valori bassi (perché si sta per entrare in stato *ready*) la media si abbassi e rientri nei range dello stato *caffè corto*

Come distinguere lo stato di off?

Dato che quando la macchina del caffè è spenta non assorbe corrente anche la presa intelligente, collegata in serie tra la macchina e la presa, non funziona più e quindi non manda più alcun valore. Quindi per distinguere tale stato si è messo un timer che controlla ogni minuto (valore temporale che potrebbe anche essere più piccolo ma per lo scopo che si voleva raggiungere va bene) se sono arrivati nuovi valori nel minuto precedente e, in caso non se ne siano ricevuti, decreta che la macchina è spenta.

4.4.1 Associare i dati ad una persona

Con il sistema presentato in questo capitolo si è in grado di monitorare il comportamento della macchina del caffè e raccogliere i dati, però manca un ultimo importante tassello, sottolineato anche nell'introduzione, l'associazione dei dati raccolti con le persone che le hanno generate.
Dal lato applicazione quanto l'utente aggiunge la macchina del caffè associa tale prodotto al proprio account. E un account, come anticipato nell'introduzione, può avere più profili.
Dall'altro lato, quando viene aggiunto il prodotto ai dispositivi controllati dal server openHAB, viene anche specificato su openHAB a quale account è associata tale macchina.
Ogni volta che su openHAB si verifica un evento, a causa del cambiamento di stato della macchina,

sul messaggio che viene inviato al sever principale che raccoglie i dati saranno mandate informazioni riguardo l'evento generato, il prodotto che lo ha generato e a quale account tale prodotto è collegato. Il server principale leggerà dal database quale profilo è attivo in quel momento su quell'account è riuscirà quindi ad associare il nuovo dato alla specifica persona (e in questo modo, ad esempio, saremo in grado di contare i caffè fatti per ciascuna persona).

Capitolo 5

CAB Product Hub

5.1 Descrizione Generale

In questa sezione si forniranno tutte le informazioni riguardo l'applicazione mobile sviluppata.
Prima di tutto verrà presentata l'intera architettura sviluppata (sezione 5.2), le motivazioni per cui è stata scelta e le sue caratteristiche, compresi i principali pattern di programmazione utilizzati.
Nella sezione use cases (sezione 5.3) si vedrà come l'utente può interagire con l'applicazione e cosa succede quando l'utente utilizza i diversi servizi presenti in essa. Successivamente, utilizzando dei sequence diagram (sezione 5.4), si analizzeranno più nel dettaglio le principali funzionalità presenti.
Poi con dei class diagram (sezione 5.5) si avrà una visuale più ampia di com'è organizzato il codice.
Infine nelle ultime tre sezioni di questo capitolo verrà presentata l'interfaccia utente (sezione 5.6) sviluppata, gli attributi (sezione 5.7) che contraddistinguono il sistema sviluppato e le principali librerie usate (sezione 5.8) all'interno dell'applicazione.

5.2 Architettura

Si è cercato di organizzare l'architettura dell'intera applicazione in blocchi tra loro il più indipendenti possibile in modo da renderla scalabile e facilmente modificabile in vista futuri aggiornamenti, magari effettuati anche da altre persone.

5.2.1 Schema generale

Il seguente grafico mostra, molto in generale, quali sono, come funzionano e in che modo sono collegati tra di loro i vari elementi che compongono il risultato finale del progetto.
Come si vede dallo schema l'utente può interagire con l'applicazione per aggiungere, connettere e gestire i tre prodotti disponibili al fine di raccogliere diversi dati (come ad esempio su come, quando e dove tali prodotti vengano usati).
L'applicazione interagisce con i prodotti ai quali si può connettere (il robot da cucina) ricevendone i dati mentre per i prodotti non tecnologici (una borsa) legge lo stato dal beacon contenuto in essi, componente dotato di tecnologia bluetooth ma non di un protocollo di connessione. L'applicazione comunica anche con il server che raccoglie i dati, tramite internet, al quale non solo invia i dati raccolti ma ne chiede a sua volta altri come i dati di login o le news.
Il prodotto tecnologico senza moduli di connessione (la macchina del caffè) non comunica, è la presa intelligente che ne legge i valori di corrente assorbiti e li invia al server su cui è installato OpenHAB. Quest'ultimo li invia al server che raccoglie i dati. I due server potrebbero anche coincidere, cioè OpenHAB potrebbe essere installato nello stesso server che si occupa di raccogliere i dati.

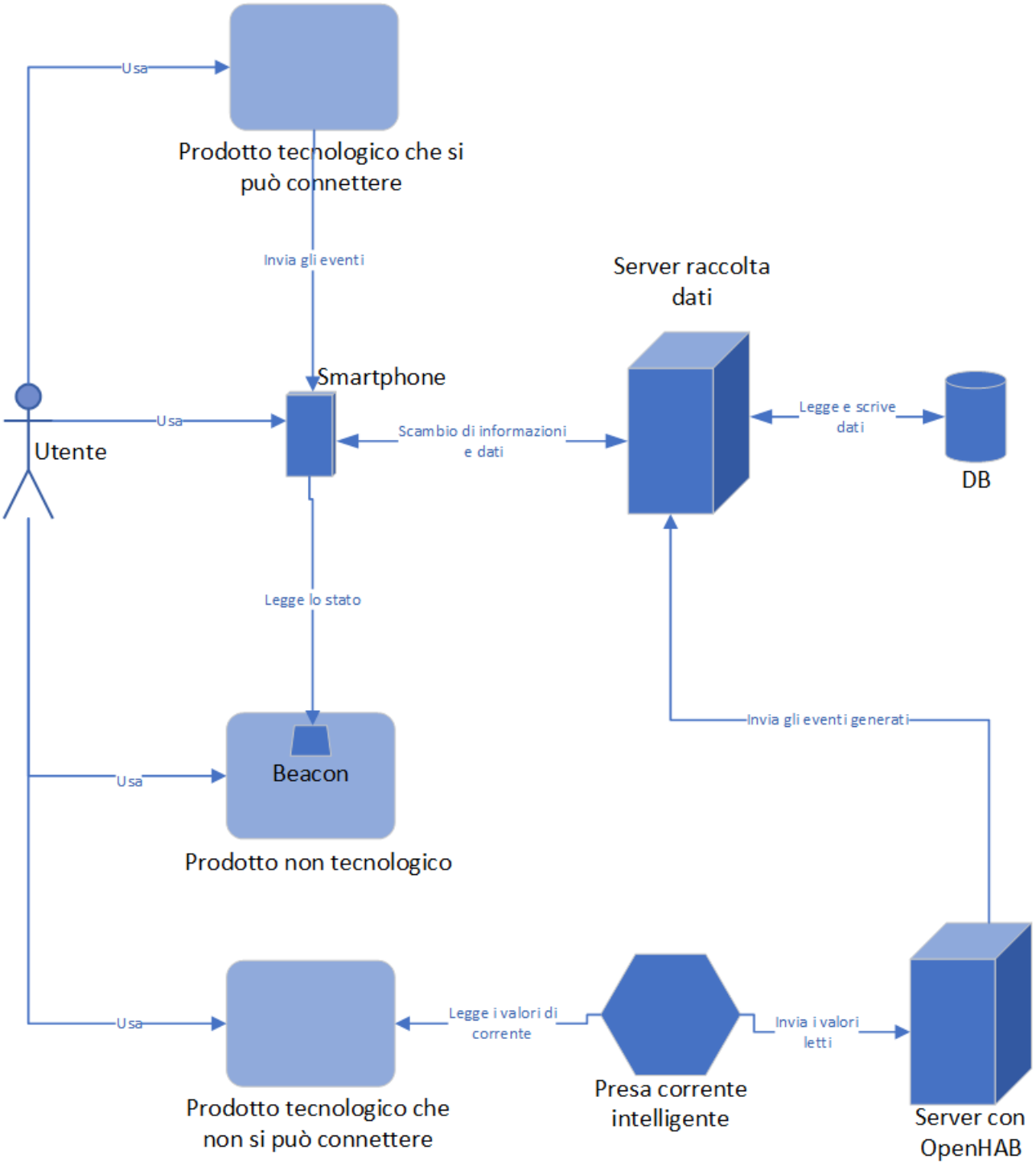


Figura 5.1: Architettura Generale

Ci si concentrerà da questo punto in poi soltanto sull'applicazione, descrivendo com'è strutturata

e le principali soluzioni che si sono implementate al suo interno.

5.2.2 Struttura a moduli

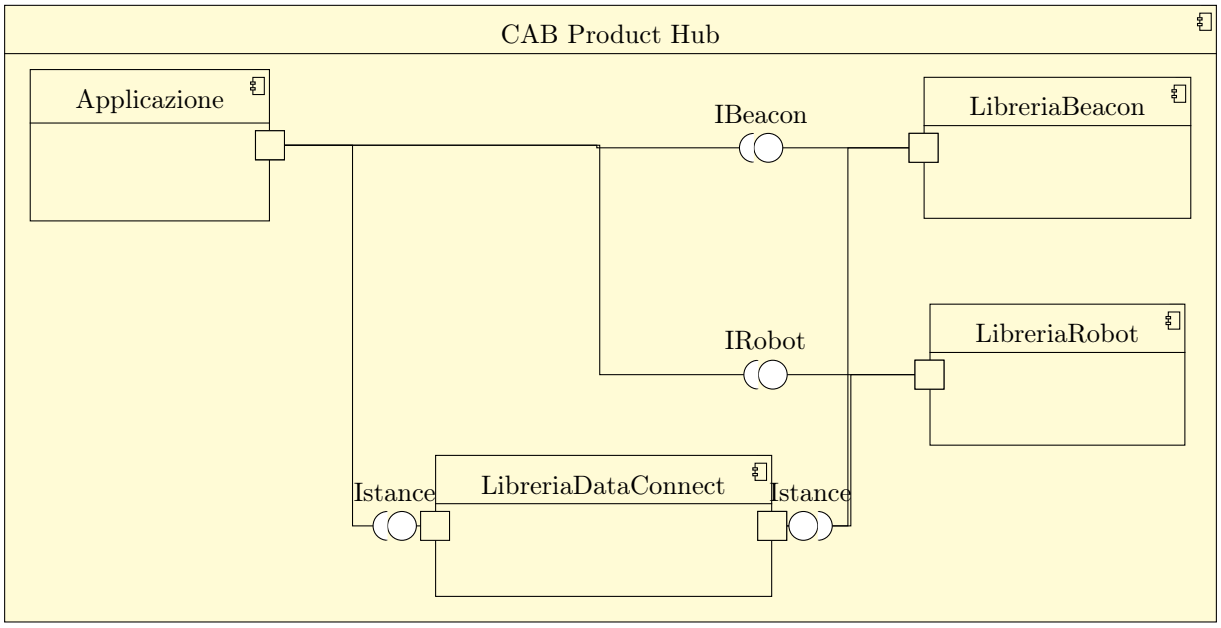


Figura 5.2: Architettura: struttura a moduli

Si è suddiviso il codice dell'applicazione in quattro diversi moduli che verranno qui presentati ma che verranno spiegati meglio, uno alla volta, nelle prossime sezioni:

- **Applicazione** rappresenta il cuore di CAB Product Hub, gestisce tutte le varie pagine presenti al suo interno in appositi fragment e contiene, ovviamente, le due activity, una per il login e registrazione e un'altra per tutto il resto. Si serve delle altre tre librerie per gestire il beacon, il robot da cucina e le comunicazioni con il server

- **Libreria Beacon** è un modulo che si occupa della completa gestione di qualsiasi beacon aggiunto dall'utente. Si connette ad esso, lo monitora ed invia gli eventi generati da esso al server utilizzando direttamente la libreria *DataConnect*. Mette a disposizione un'interfaccia che rende pubblici soltanto i metodi necessari all'applicazione che la usa tenendo così nascosta l'implementazione

- **Libreria Robot** è un modulo che si occupa della completa gestione di qualsiasi robot da cucina, compatibile, aggiunto dall'utente. Si connette ad esso, lo monitora ed invia gli eventi generati

da esso al server utilizzando direttamente la libreria *DataConnect*. Anch'essa mette a disposizione un'interfaccia che rende pubblici i metodi necessari all'applicazione che la usa tenendo così nascosta l'implementazione.

- **Libreria DataConnect** è un modulo che gestisce tutte le richieste, sia *POST* che *GET*, fatte al server da qualsiasi punto dell'applicazione. Viene utilizzata anche dalle altre librerie incluse nell'applicazione e mette a disposizione un'istanza tramite la quale è possibile chiamare i suoi metodi pubblici da qualsiasi punto dell'applicazione (*pattern Singleton*) senza bisogno di istanziarla svariate volte in diversi punti dell'applicazione.

Sviluppare il codice in moduli rende il codice più comprensibile diminuendo la probabilità di introdurre errori e rendendo più semplice il testing. Inoltre semplifica futuri aggiornamenti dato che il programmatore potrà concentrarsi solo su una porzione di codice ignorando, quasi completamente, gli altri moduli.
Oltre a questi vantaggi si è suddiviso il codice anche per motivi commerciali. Lavorando con altre aziende se una di queste volesse sviluppare una propria applicazione che gestisca solo i beacon gli si potrà fornire soltanto la libreria dei beacon invece dell'intera applicazione.

Oscuramento del codice

Le tre librerie usate hanno tutte il codice oscurato per motivi di sicurezza. In questo modo si potrebbe condividere il codice dell'applicazione con un'azienda partner senza però dare dettagli sulle librerie.
Un altro esempio, per maggiore chiarezza, riguarda la libreria del robot da cucina. Se una terza azienda vuole raccogliere dati su esso potrà farlo usando l'apposita libreria che terrà oscurato il protocollo con cui avviene la connessione con la pentola.

5.2.3 Libreria beacon

Si è sviluppata e testata questa libreria per la gestione dei beacon in un'applicazione esterna a CAB Product Hube successivamente inserita.
Il monitoraggio dei beacon, per sapere se sono nei d'intorni e se si stanno muovendo, avviene attraverso

la libreria Estimote (5.8.4), creata dalla stessa azienda che produce i beacon. Oltre a questo la libreria dei beacon offre un'interfaccia, legge i dati della posizione dal GPS, gestisce l'invio degli eventi ed è scritta per risparmiare, per quanto possibile, la batteria dello smartphone.

Monitoraggio beacon

Attraverso la libreria Estimote viene letto lo stato dei beacon che si trovano nel raggio di funzionamento del bluetooth.
Identifichiamo quattro differenti stati per un beacon:

- Il beacon è in movimento (*moving*)
- Il beacon è fermo (*not moving*)
- Il beacon è stato allontanato troppo dallo smartphone (*lost*)
- Il beacon è stato trovato (*found*)

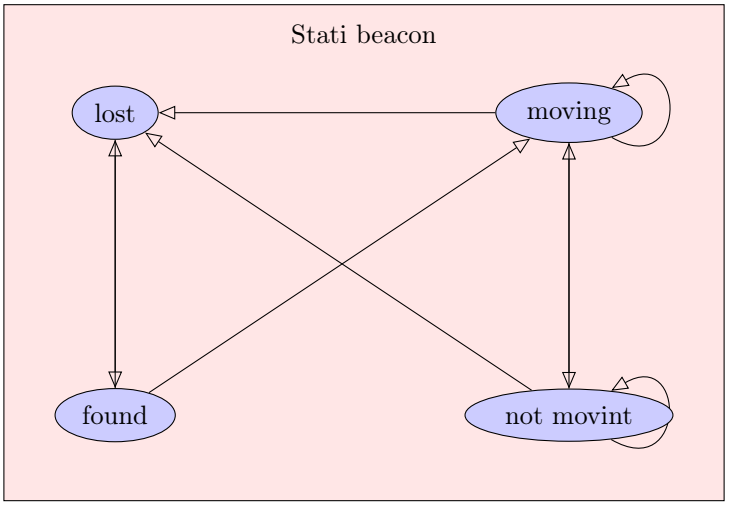


Figura 5.3: Architettura: Stati del beacon

Interfaccia

Viene utilizzata un interfaccia che fornisce all'applicazione in cui viene usata una serie di metodi pubblici che permettono:

- Di settare l'URL del server al quale si vogliono mandare gli eventi, con il relativo token necessario all'autenticazione durante la richiesta POST al server
- Inviare e rimuovere un beacon
- Abilitare o disabilitare l'invio degli eventi o solo della posizione degli eventi
- Leggere l'ultimo evento generato
- Per gli sviluppatori è possibile settare ogni quanti minuti inviare gli eventi generati e accumulati nel database gestito dalla libreria al server

Lettura dati del GPS

Una volta dati i permessi all'applicazione (la cui gestione è lasciata all'applicazione che usa la libreria in quanto richiede di un'activity) e attivato il GPS sullo smartphone, la libreria è in grado di leggere la posizione corrente usando i servizi di Google. Tale posizione, se l'utente ne dà il permesso nella pagina dedicata al beacon, viene inviato al server insieme all'evento generato.

Gestione intelligente dell'invio degli eventi

Ogniqualvolta il beacon cambia stato si genera un evento che viene salvato, insieme alla posizione, in un database gestito dall'applicazione stessa.
Ogni ora, valore utilizzato nei test ma che potrebbe cambiare senza problemi, tutti gli eventi generati vengono inseriti in un array di oggetti JSON e, se l'utente ne ha abilitati i permessi, inviati tutti al server utilizzando la libreria che gestisce la connessione con il server. In questo modo al posto di effettuare tante piccole connessioni che inviano pochi dati se ne effettuano poche che ne inviano di più rendendo il tutto più scalabile.

Risparmio energetico

Grazie alla libreria Estimote, che permette di configurare ogni quanti secondi effettuare la lettura dello stato del beacon e per quanti secondi provare a leggere lo stato, si può gestire in maniera intelligente l'utilizzo del bluetooth in modo da evitare un consumo eccessivo della batteria.

Dopo vari esperimenti si è trovato il giusto compromesso tra precisione e risparmio energetico usando 2 secondi per la scansione e 20 secondi per l'attesa tra una scansione e l'altra. Con *scansione* si intende quell'intervallo di tempo in cui il bluetooth rimane attivo e usato per rilevare e leggere lo stato beacon. Sotto è visibile un sequence diagram esemplificativo su come funziona tale procedimento.

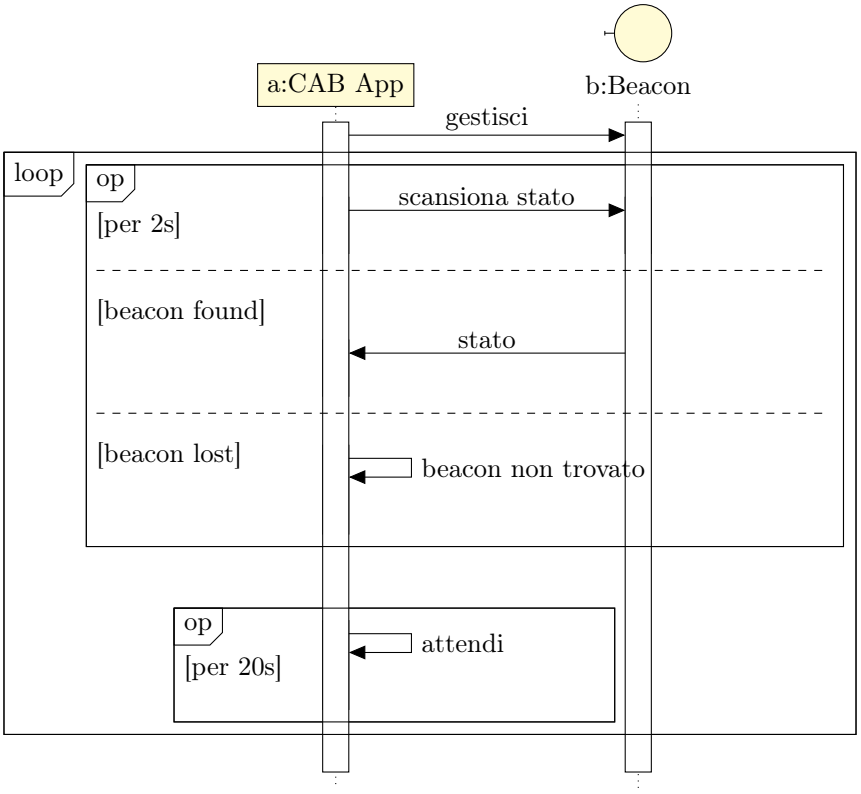


Figura 5.4: Libreria becon: gestione rilevazione beacon

Un servizio sempre attivo

In questa libreria, come nella successiva che gestisce il robot da cucina, la classe che monitora il beacon è stata sviluppata estendendo la classe *Service*. Questo permette di avere un servizio sempre attivo che in Android vuol dire poter chiudere l'applicazione senza che il servizio venga chiuso.
In questo modo l'utente non deve tenere perennemente aperta l'applicazione per raccogliere dati, cosa che previene anche il consumo eccessivo della batteria.

5.2.4 Libreria robot da cucina

Non si descriverà nei dettagli la libreria che gestisce il robot da cucina perché non è stata implementata durante il progetto ma soltanto usata. Tuttavia il suo funzionamento è molto simile alla libreria dei beacon, fornendo una analoga interfaccia all'applicazione che la usa.

5.2.5 Libreria DataConnect

Anche questa libreria è stata sviluppata e testata in un'applicazione esterna a CAB Product Hub. Come già detto sopra si occupa di gestire le comunicazioni con il server e viene utilizzata anche dalle altre librerie.
Tramite un *singleton* dà l'accesso ai vari metodi che permettono di:

- Leggere le news
- Ricevere i prodotti che l'applicazione può supportare in base alla versione
- Ricevere, aggiungere e rimuovere i prodotti associati ad un utente
- Inviare i dati raccolti dai prodotti al server

Si accede al singleton chiamando un metodo statico della classe al quale vengono passati i seguenti campi:

- Il *context dell'activity*, necessario per istanziare ed usare la libreria Volley
- L'*indirizzo del server* con il quale l'applicazione vuole connettersi
- Un *token* che verrà utilizzato per effettuare l'autenticazione quando si effettua una richiesta al server

Si è utilizzato un singleton perché la classe principale della libreria che gestisce la connessione viene chiamata da più punti all'interno dell'applicazione e usando un singleton non è necessario istanziarla ogni volta o creare un istanza fissa per ogni classe.

Gestione degli errori di rete

Gli errori di rete vengono affrontati provando ad inviare le richieste per tre volte, con un tempo di timeout di 3 secondi e salvando le richieste che falliscono tutti i tentativi per problemi di rete in un

proprio database, gestito dalla libreria stessa. Successivamente si riprova ad inviare le richieste ogni ora (intervallo di tempo che può ovviamente essere cambiato). Se la richiesta viene inviata con successo la si elimina dal database mentre se fallisce nuovamente viene aggiornato un contatore di tentativi associato a tale richiesta.
Per evitare che il database si riempa al punto da far occupare all'applicazione troppa memoria sullo smartphone, la libreria alla fine di ogni tentativo di rinvio delle richieste fallite fa un controllo di tutto il database eliminando tutte le richieste il cui invio è fallito per 3 volte di fila (anch'esso valore che può essere modificato).
Come ultimo accorgimento, al fine di non inviare troppe richieste al server nello stesso istante, ad ogni tentativo di invio di tutte le richieste fallite in precedenza si dà un ritardo tra una richiesta e l'altra di un secondo.
Guardare nel capitolo dei sequence diagram (5.4.7) per vedere lo schema grafico di come funziona la gestione degli errori.

5.2.6 Applicazione principale

Nell'applicazione principale, che di fatto rappresenta CAB Product Hub, vengono utilizzate tutte le librerie viste in precedenza per gestire sia vari dispositivi che si possono connettere con l'applicazione che le comunicazioni con il server.
Per organizzare l'applicazione si è utilizzata la suddivisione tipica di tutte le applicazioni android, cioè in activity, ognuna delle quali gestisce differenti fragment.
Più nello specifico è stata suddivisa in due principali activity:

- La prima gestisce i due fragment di login e la registrazione

- La seconda gestisce tutto il resto dell'applicazione, con differenti fragment, uno per ogni pagina

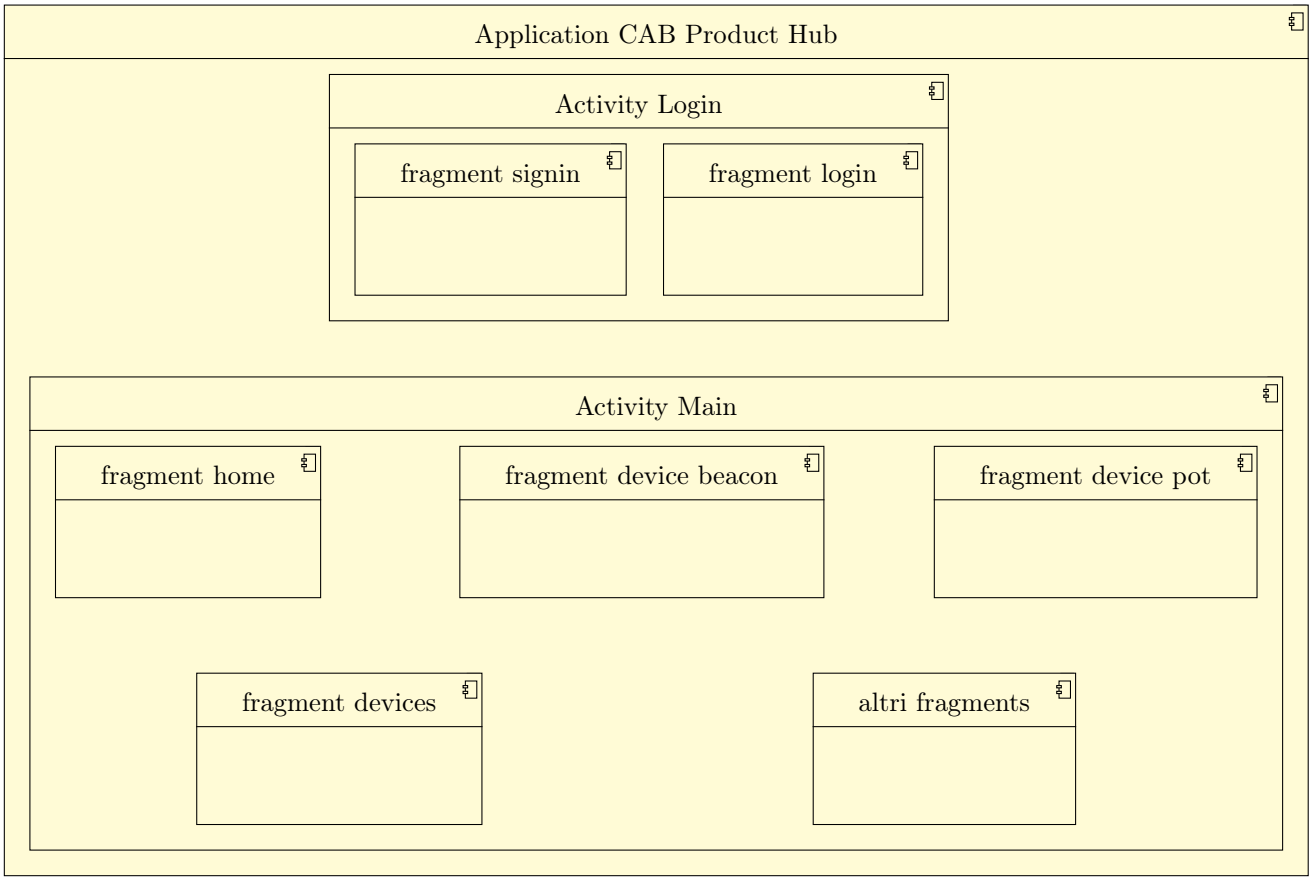


Figura 5.5: Schema struttura dell'applicazione (nella Activity Main non sono stati elencati tutti i fragment che la compongono ma solo i principali)

Nell'application sono istanziati la libreria *Estimote* utilizzata per la gestione dei beacon e una istanza di Volley usata come caricatore di immagini online.

Nelle activity sono racchiusi tutti quei metodi non direttamente connessi ad una singola pagina dell'applicazione ma che possono essere utilizzati da più pagine, o da nessuna, come:

- *Gestione dei permessi* concessi all'applicazione, come il permesso di usare il bluetooth o il GPS dello smartphone
- Monitoraggio dello *stato della connessione* attraverso un listener che si attiva ogni volta che lo stato di rete del cellulare cambia, facendo apparire una barra rossa in basso che avverte l'utente

di non essere più connesso ad internet, con conseguente diminuzione delle funzionalità disponibili nell'app

- Gestione del *menu a comparsa* laterale che, presente in tutte le pagine, permette di navigare all'interno di CAB Product Hub
- Una serie di metodi, uno per ogni fragment, che permettono di impostare un nuovo fragment attivo ogni volta che si vuole cambiare pagina durante la navigazione
- Un metodo che ritorna lo *stato del GPS*

Ciascun fragment si occupa di gestire le azioni che un utente può fare nella relativa pagina. Non si starà qui ad elencare tutti i fragment perché nel capitolo 5.6 si presenteranno tutte le pagine dell'applicazione e, sapendo che esiste un fragment per ciascuna pagina, verranno di fatto elencati anche tutti i fragment.

Una raccolta di "utils" suddivise in diverse classi contengono metodi e variabili utilizzati in diversi punti dell'applicazione, in particolare un file che contiene tutte le *principali costanti* usate nell'applicazione e un altro che gestisce le *preferenze*.

Osservazioni importanti sull'applicazione

- **Bluetooth e GPS**

  Perché bisogna attivare il GPS per connettersi con il robot da cucina o il beacon?
  La causa è dovuta alla libreria android che gestisce il bluetooth, la quale per far funzionare il metodo che fa la scansione di tutti i dispositivi bluetooth nei dintorni necessita del GPS attivo.
  Tuttavia la posizione dell'utente non verrà inviata a meno che non sia lui stesso ad abilitarne l'invio nei permessi del device.

- **Riavvio al reboot**

  I servizi che gestiscono sia il beacon che il robot da cucina ricevendo gli eventi ed inviandoli al server vengono automaticamente farti ripartire all'accensione dello smartphone se erano già attivi prima che lo stesso venisse spento.

5.2.7 Gestione di account e profili

Quando l'utente si iscrive per la prima volta crea un account e potrà usare i dati dell'account per accedere all'applicazione. Poi all'interno del suo account potrà creare differenti profili, ad esempio uno per ogni membro della sua famiglia.
Quando un prodotto viene aggiunto dall'utente tale prodotto verrà associato al suo account in maniera univoca e uno stesso prodotto non potrà essere aggiunto da altri account.
L'utente prima di iniziare la raccolta dadi selezionerà un profilo, che verrà settato come principale, e i dati raccolti verranno associati a tale profilo.
L'account viene visto nel sistema come il "proprietario" del prodotto mentre i profili come le persone che lo usano. In questo modo il prodotto non deve essere aggiunto da più account.

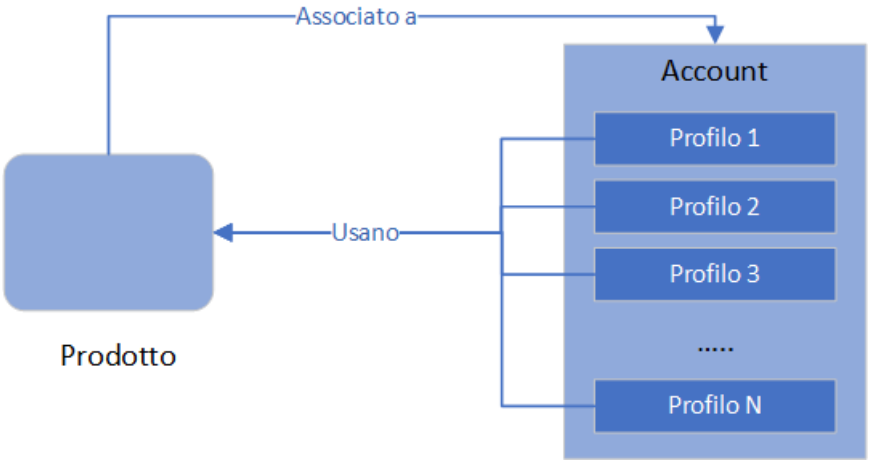


Figura 5.6: Account e profili

Esempio pratico

Una famiglia riceve il robot da cucina per testarlo e raccoglierne dati. Il padre installa l'applicazione sul proprio smartphone, si crea un account, aggiunge il robot da cucina e aggiunge un profilo per ogni membro della propria famiglia. Quando un membro della famiglia userà il robot da cucina setterà il proprio profilo come attivo usando lo smartphone del padre o scaricandosi l'applicazione e accedendo con gli stessi dati del padre.

NFC

Cambiare profilo ogni volta che una persona diversa utilizza un prodotto è un metodo efficace finché si è in un piccolo contesto, ad esempio familiare. Ma se si stanno raccogliendo dati in un ufficio con parecchi dipendenti dover cambiare manualmente il profilo ogni volta che un dipendente fa un caffè può diventare un'operazione lenta e sgradevole.
Per ovviare a questo problema è stato introdotto nell'applicazione la possibilità di cambiare profilo avvicinando all'applicazione un badge dotato di tecnologia NFC (stessa tecnologia di cui sono provvisti anche gli smartphone).

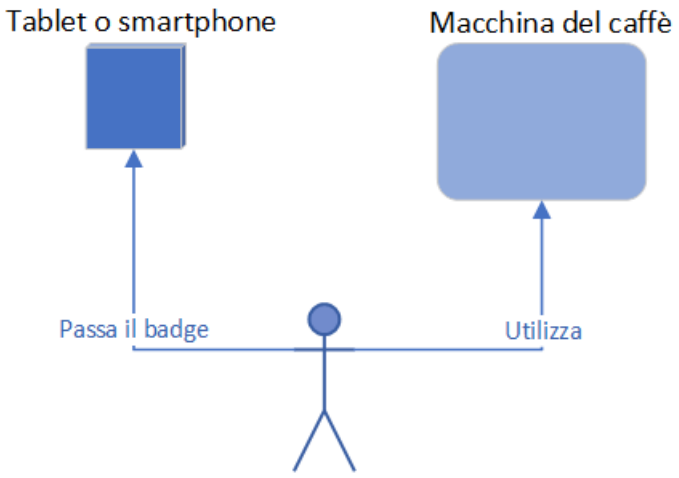


Figura 5.7: Cambiare profilo col badge

In questo modo viene posto un tablet (o uno smartphone), con l'applicazione attiva, di fianco alla macchina del caffè. Si sarà effettuato l'accesso con l'account al quale è associato la macchina e si dovranno creare tanti profili quante sono le persone che usufruiscono della macchina. Sul server per ciascun profilo si dovrà fornire il corrispettivo codice badge.
Quando qualcuno nell'ufficio vorrà utilizzare la macchina del caffè dovrà avvicinare il proprio badge al tablet. L'applicazione, attraverso l'NFC dello smartphone, leggerà il dal badge e cambierà il profilo attivo con quello a cui è associato il codice letto.
Durante il progetto tale funzionalità è stata inserita nell'applicazione ma non testata in un contesto reale.
Inoltre nell'applicazione il badge viene aggiunto e salvato come identifier di uno specifico profilo. Si

è utilizzato il termine generico "identifier" (che vuol dire *identificatore*) perché si è ipotizzato che in futuro possano essere aggiunti altri dispositivi per identificare un certo profilo.

5.2.8 Pattern utilizzati

Vediamo in questa sezione i principali pattern adottati nell'applicazione.[20]

Singleton

Il pattern *Singleton* permette di assicurare che ci sia una sola istanza di una certa classe.
È stato usato questo pattern per avere un'unica istanza della classe che gestisce le comunicazioni con il server, dato che viene utilizzata da diverse altre classi presenti nell'applicazione.

Bridge

Il pattern *Bridge* permette di separare l'interfaccia di una classe, cioè che cosa si può fare con quella classe, dalla sua implementazione, cioè come la si fa. Detto in altre parole è possibile far vedere a chi usa la classe soltanto i metodi che può effettivamente utilizzare, tenendo oscurati quelli usati per implementare l'intero funzionamento della classe.
È stato usato questo pattern nelle librerie per il beacon e per il robot da cucina per fornire un'interfaccia di tutti i metodi della libreria che possono essere utilizzati, nascondendone altri usati per l'implementazione dell'intero servizio.

Template

Il pattern *Template* permette di definire la struttura di una classe con metodi comuni alle classi che la erediteranno in modo da non dover riscrivere lo stesso codice più volte.
Si è usato questo pattern per creare una classe contenente i metodi comuni a tutti i fragment e activity presenti nell'applicazione. In questo modo si è inoltre sicuri che ciascun fragment e activity contenga tali metodi che possono poi essere sovrascritti se necessario.
A scopo di esempio un metodo presente in tutti i fragment è il metodo *onBack* che permette di tornare alla pagina precedente.[21]

5.3 Use Cases

In questa parte del documento si vedrà come l'utente può interagire con l'applicazione per utilizzare i diversi servizi offerti, senza però vedere come funziona il sistema al suo interno.
Si può identificare un solo attore coinvolto nell'applicazione:

- *utente* - colui che ha scaricato e installato CAB Product Hub

Per le funzionalità più importanti o complesse oltre alla tabella è riportato un schema grafico.

5.3.1 Effettuare la registrazione

Si nota che tale funzionalità è stata rimossa per scelte dell'azienda partner che non ha fornito le API necessarie rimandando l'utente sul loro sito per la registrazione.

Attore	Utente
Obiettivo	Effettuare la registrazione a CAB Product Hub
Flusso di Eventi	Premere "No account yet? Create one" nella pagina iniziale Inserire i dati richiesti Premere "sign in"
Condizione di Ingresso	Aver scaricato l'app
Condizione di Uscita	Viene creato il nuovo account nel database del server
Eccezione	L'utente ha lasciato qualche campo vuoto L'utente ha sbagliato a scrivere la password di conferma

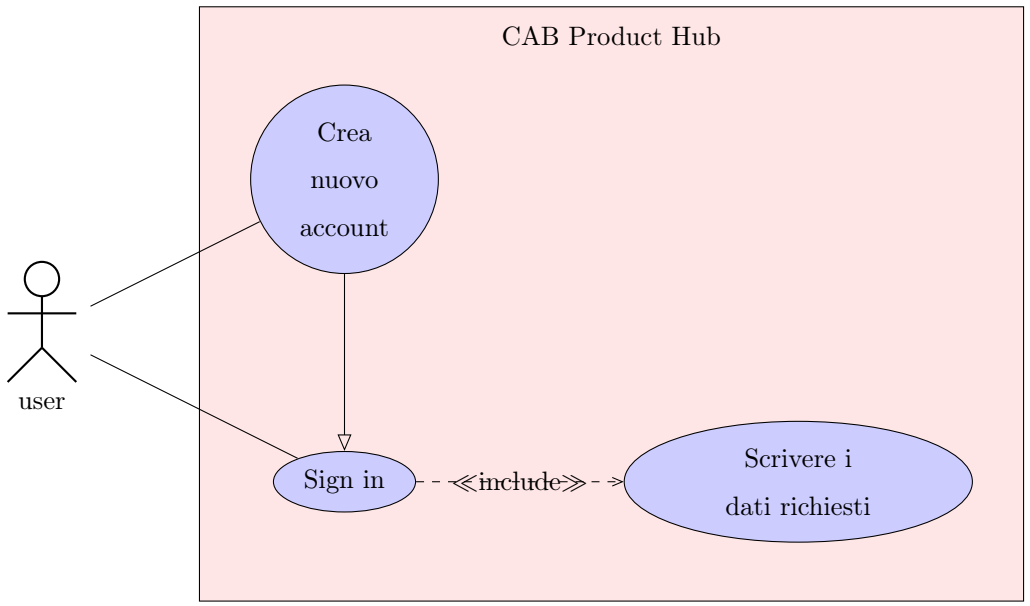


Figura 5.8: Use Case: Effettuare la registrazione

5.3.2 Effettuare il login

Se non ha fatto il logout l'utente non ha la necessità di effettuare il login ogni volta che apre l'applicazione ma solo la prima volta o quando il token, creato durante un login precedente, scade.

Attore	Utente
Obiettivo	Effettuare il login per accedere a CAB Product Hub
Flusso di Eventi	Inserire email e password con i quali ci si è registrati Premere "login"
Condizione di Ingresso	Aver già effettuato la registrazione Token non presente o scaduto
Condizione di Uscita	Login effettuato e visualizzazione della home page dell'applicazione
Eccezione	L'utente ha sbagliato ad inserire i propri dati d'accesso L'utente non è registrato

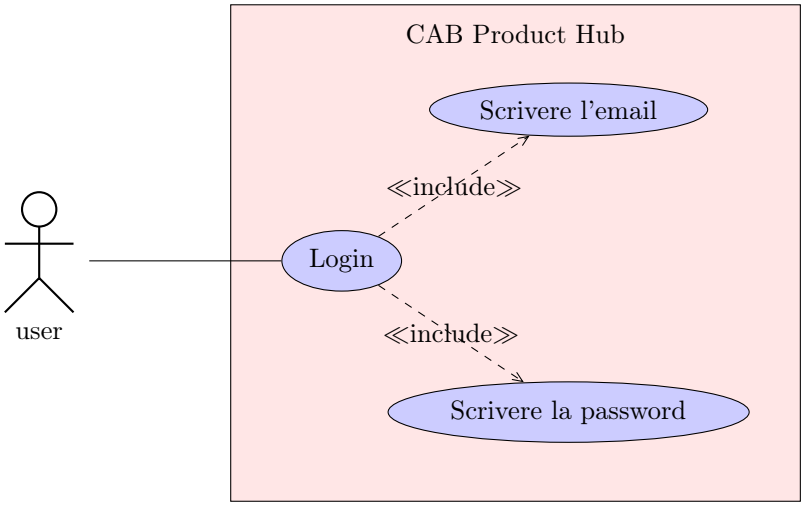


Figura 5.9: Use Case: Effettuare il login

5.3.3 Aggiungere un nuovo beacon

Attore	Utente
Obiettivo	Aggiungere un nuovo beacon tra i propri dispositivi
Flusso di Eventi	Nel menu a scomparsa selezionare la voce "Devices" Premere il pulsante rosso in basso a sinistra Attivare il GPS (se non lo si è già fatto) Selezionare il beacon desiderato nella lista dei dispositivi compatibili con la versione della propria app Dare i permessi per utilizzare il bluetooth se è la prima volta che ci si collega ad un dispositivo utilizzando CAB Product Hub Nella lista di tutti i beacon rilevati selezionare il proprio
Condizione di Ingresso	Aver già effettuato il login Che il beacon che si vuole aggiungere sia supportato
Condizione di Uscita	Beacon aggiunto ai propri device, si va nella pagina personale del beacon e viene effettuata la connessione
Eccezione	L'utente non ha dato i permessi per utilizzare il bluetooth L'utente non ha attivato il gps

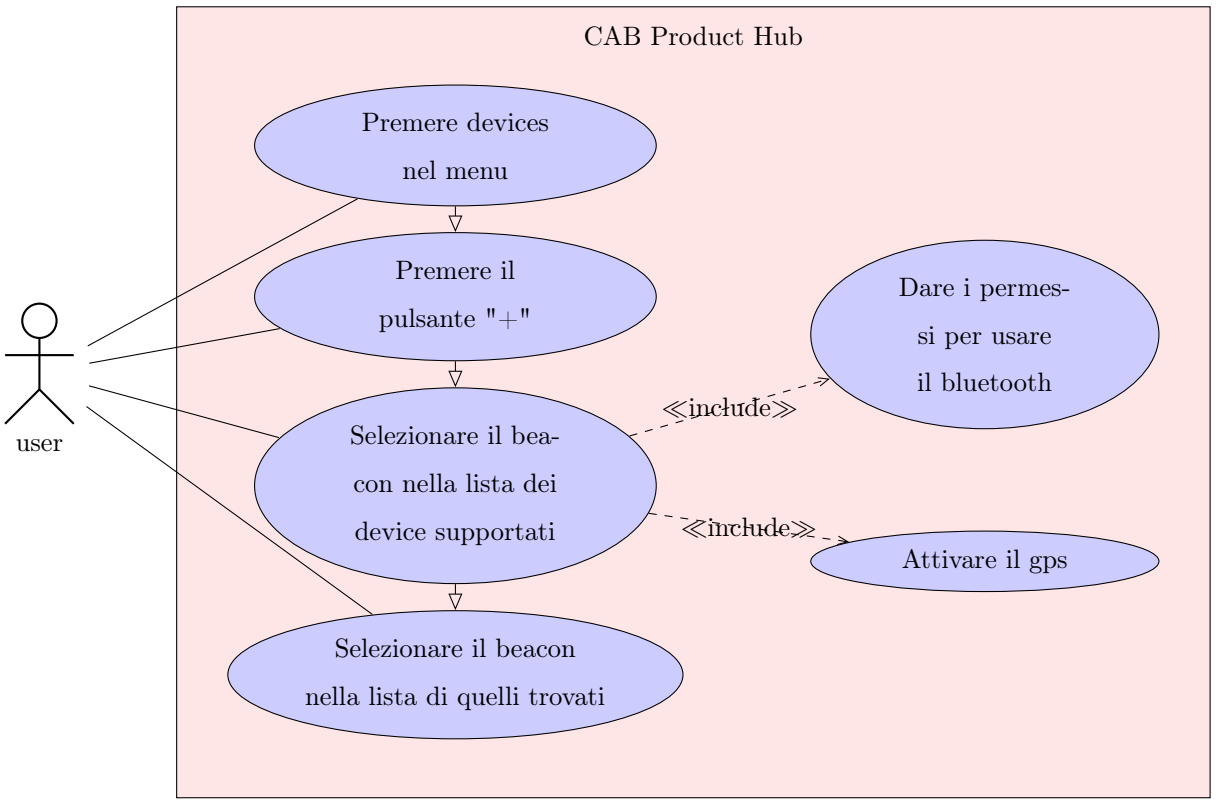


Figura 5.10: Use Case: Aggiungere un nuovo beacon

5.3.4 Aggiungere un nuovo robot da cucina

Attore	Utente
Obiettivo	Aggiungere un nuovo robot da cucina tra i propri dispositivi
Flusso di Eventi	Nel menu a scomparsa selezionare la voce "Devices" Premere il pulsante rosso in basso a sinistra Attivare il GPS (se non lo si è già fatto) Selezionare il robot da cucina desiderato nella lista dei dispositivi compatibili con la versione della propria app Dare i permessi per utilizzare il bluetooth se è la prima volta che ci si collega ad un dispositivo utilizzando CAB Product Hub Nella lista di tutti i dispositivi rilevati dal bluetooth selezionare il robot da cucina (distinguibile per il nome)
Condizione di Ingresso	Aver già effettuato il login Che il robot da cucina che si vuole aggiungere sia supportato
Condizione di Uscita	Nuovo device aggiunto ai propri, si va nella pagina personale del robot da cucina e viene effettuata la connessione
Eccezione	L'utente non ha dato i permessi per utilizzare il bluetooth L'utente non ha attivato il gps L'utente ha selezionato il dispositivo sbagliato nella lista dei dispositivi rilevati dal bluetooth

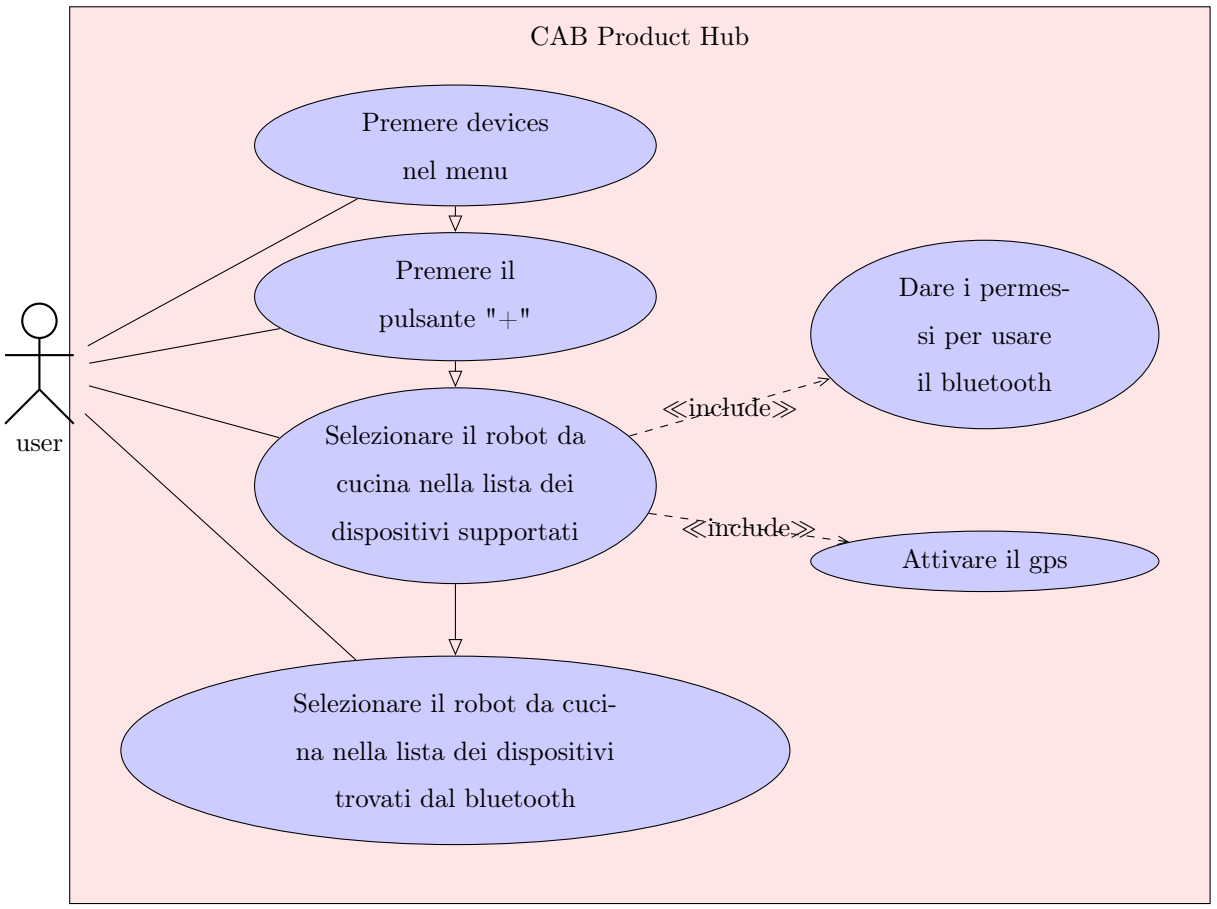


Figura 5.11: Use Case: Aggiungere un nuovo robot da cucina

5.3.5 Aggiungere una nuova macchina del caffè

A differenza dei precedenti, la macchina del caffè non si connette direttamente allo smartphone ma l'utente può aggiungerla per gestirne i permessi.

Attore	Utente
Obiettivo	Aggiungere una nuova macchina del caffè tra i propri dispositivi
Flusso di Eventi	Nel menu a scomparsa selezionare la voce "Devices" Premere il pulsante rosso in basso a sinistra Selezionare la macchina del caffè nella lista dei dispositivi compatibili con la versione della propria app
Condizione di Ingresso	Aver già effettuato il login
Condizione di Uscita	Nuovo device aggiunto ai propri e si va nella pagina personale della macchina del caffè
Eccezione	-

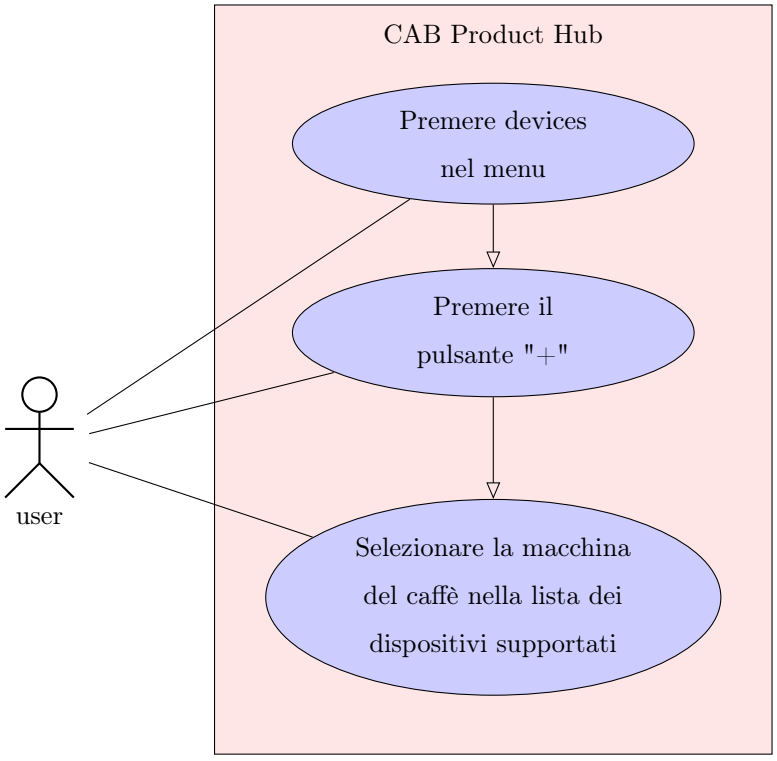


Figura 5.12: Use Case: Aggiungere una nuova macchina del caffè

5.3.6 Connettersi ad un beacon

Attore	Utente
Obiettivo	Connettersi ad un beacon precedentemente aggiunto
Flusso di Eventi	Nel menu a scomparsa selezionare la voce "Devices" Attivare il GPS (se non lo si è già fatto) Selezionare il beacon desiderato nella lista dei dispositivi aggiunti
Condizione di Ingresso	Aver già aggiunto il beacon
Condizione di Uscita	Si va nella pagina personale del beacon e viene effettuata la connessione
Eccezione	L'utente non ha attivato il gps

5.3.7 Connettersi ad un robot da cucina

Attore	Utente
Obiettivo	Connettersi ad un robot da cucina precedentemente aggiunto
Flusso di Eventi	Nel menu a scomparsa selezionare la voce "Devices" Attivare il GPS (se non lo si è già fatto) Selezionare il robot da cucina desiderato nella lista dei dispositivi aggiunti
Condizione di Ingresso	Aver già aggiunto il robot da cucina
Condizione di Uscita	Si va nella pagina personale del robot da cucina e viene effettuata la connessione
Eccezione	L'utente non ha attivato il gps

5.3.8 Modificare i permessi di un device connesso

I permessi che si possono abilitare (o disabilitare) dipendono dal dispositivo perché differenti dispositivi raccolgono informazioni differenti. Tuttavia l'abilitazione (o disabilitazione) funziona in maniera sempre uguale in tutti i device.

Attore	Utente
Obiettivo	Modificare i permessi di un device connesso
Flusso di Eventi	Abilitare o disabilitare lo switch di fianco ad "Enable sending usage information" per permettere (o impedire) all'applicazione di mandare informazioni riguardo l'attività dell'utente Abilitare o disabilitare lo switch di fianco ad "Enable sendig your location" per permettere (o impedire) all'applicazione di includere la propria posizione nelle informazioni inviate dall'applicazione
Condizione di Ingresso	Essere nella pagina personale di un device
Condizione di Uscita	Si cambiano i permessi di un device
Eccezione	-

5.3.9 Visualizzare la macchina del caffè

Attore	Utente
Obiettivo	Visualizzare la pagina dedicata alla macchina del caffè
Flusso di Eventi	Nel menu a scomparsa selezionare la voce "Devices" Selezionare il la macchina del caffè nella lista dei device aggiunti
Condizione di Ingresso	Aver già aggiunto la macchina del caffè
Condizione di Uscita	Si va nella pagina personale della macchina del caffè
Eccezione	-

5.3.10 Aggiungere un nuovo profilo

Attore	Utente
Obiettivo	Aggiungere un nuovo profilo
Flusso di Eventi	Nel menu a scomparsa selezionare la voce "Profiles" Premere il pulsante "+" in basso a destra Inserire il nickname Premere "ok"
Condizione di Ingresso	Aver effettuato il login
Condizione di Uscita	Viene aggiunto un nuovo profilo alla lista
Eccezione	Il nickname che si è inserito esiste già

5.3.11 Cambiare profilo attivo manualmente

Attore	Utente
Obiettivo	Cambiare profilo attivo manualmente
Flusso di Eventi	Nel menu a scomparsa selezionare la voce "Profiles" Dalla lista cliccare sul profilo che si vuole rendere attivo Premere il bottone rosso con la scritta "Log events as me"
Condizione di Ingresso	Aver già aggiunto il profilo che si vuole rendere attivo
Condizione di Uscita	Il profilo selezionato è adesso attivo
Eccezione	-

5.3.12 Aggiungere un badge ad un profilo

Attore	Utente
Obiettivo	Aggiungere un badge ad un profilo
Flusso di Eventi	Nel menu a scomparsa selezionare la voce "Profiles" Cliccare sul profilo sul quale si vuole aggiungere il nuovo badge Cliccare sul pulsante "+" in basso a destra Inserire un nickname per ricordarsi di che identifier si tratta Poggiare il badge allo smartphone finché non viene letto il codice Premere su "ok"
Condizione di Ingresso	Aver creato il profilo al quale si vuole aggiungere il badge
Condizione di Uscita	Nuovo badge aggiunto
Eccezione	badge già esistente NFC spento

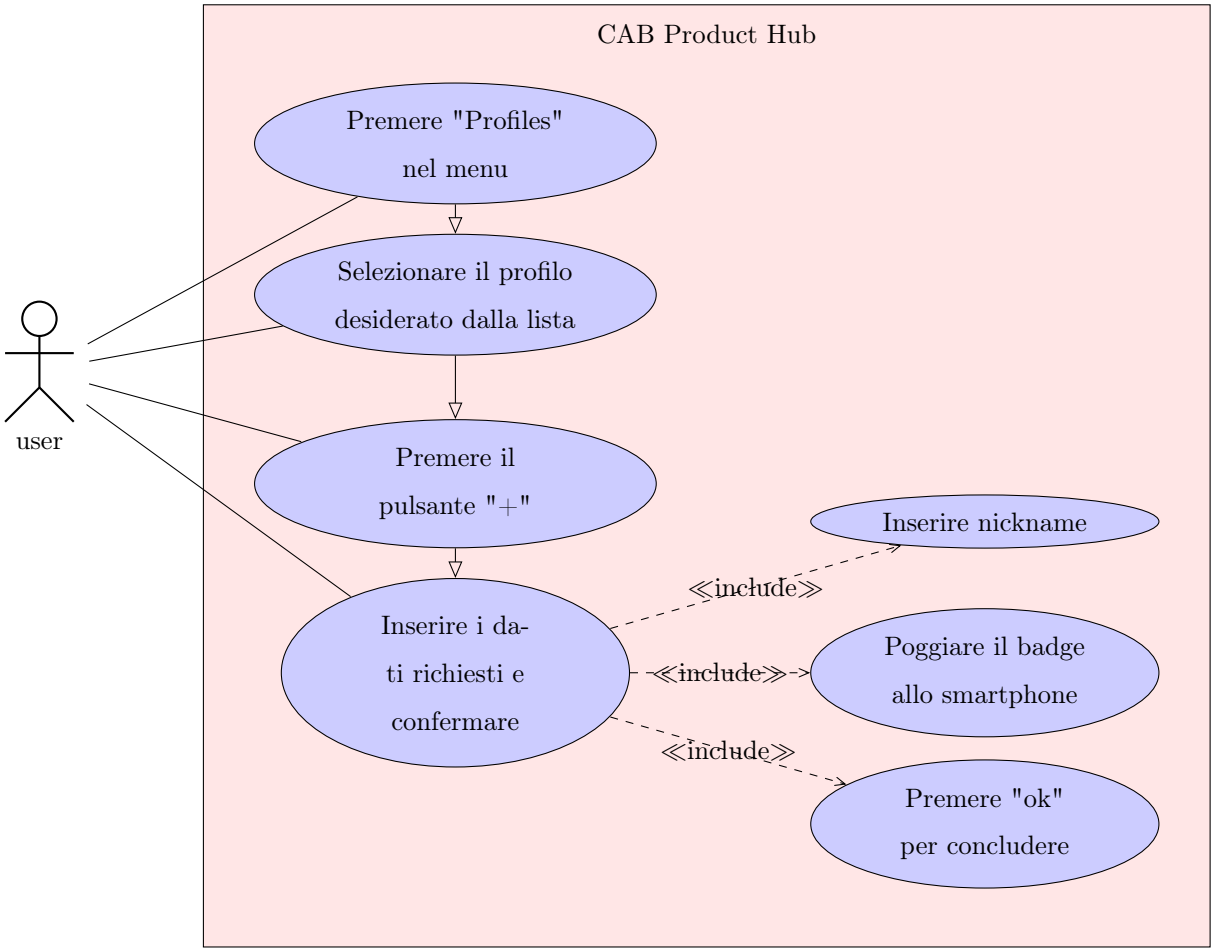


Figura 5.13: Use Case: Aggiungere un nuovo badge

5.3.13 Cambiare profilo attivo tramite badge

Attore	Utente
Obiettivo	Cambiare profilo per la raccolta dati tramite badge
Flusso di Eventi	poggiare il badge allo smartphone e tenerlo finché non apparirà la conferma di aver cambiato il profilo attivo
Condizione di Ingresso	Che il profilo esista e sia stato aggiunto nella sua lista di identifier il badge che si sta usando
Condizione di Uscita	Il profilo attivo è cambiato
Eccezione	Al badge che si sta usando non è associato alcun profilo

5.3.14 Visualizzare le news

Attore	Utente
Obiettivo	Visualizzare le news
Flusso di Eventi	Nel menu a scomparsa selezionare la voce "Home"
Condizione di Ingresso	Aver effettuato il login
Condizione di Uscita	Si va nella pagina iniziale dove è visualizzabile un elenco di news
Eccezione	-

5.3.15 Visualizzare gli analytics

Attore	Utente
Obiettivo	Visualizzare la pagina degli Analytics
Flusso di Eventi	Nel menu a scomparsa selezionare la voce "Analytics"
Condizione di Ingresso	Aver effettuato il login
Condizione di Uscita	Si va nella pagina degli Analytics
Eccezione	-

5.3.16 Visualizzare le impostazioni

Attore	Utente
Obiettivo	Visualizzare la pagina delle impostazioni
Flusso di Eventi	Nel menu a scomparsa selezionare la voce "Settings"
Condizione di Ingresso	Aver effettuato il login
Condizione di Uscita	Si va nella pagina dei Settings
Eccezione	-

5.3.17 Visualizzare i settings per sviluppatori

Sono delle opzioni particolari create a scopo di debug e normalmente non visibili.

Attore	Utente
Obiettivo	Visualizzare la pagina delle impostazioni per sviluppatori
Flusso di Eventi	Nella pagina delle impostazioni fare doppio click sulla versione dell'app
Condizione di Ingresso	Trovarsi nella pagina delle impostazioni
Condizione di Uscita	Vengono rese visibili le impostazioni segrete per sviluppatori
Eccezione	Effettuare il doppio click lentamente

5.4 Sequence Diagrams

In questo capitolo si vedranno più nel dettaglio, attraverso appunto dei sequence diagrams, come sono implementate le funzionalità più complesse presenti nell'applicazione.
Alcune osservazioni per comprendere al meglio i sequence diagram:

- I rettangoli colorati di verde contengono per operazioni svolte dall'applicazione se lo smartphone è connesso ad internet

- allo stesso modo i rettangoli blu contengono operazioni che vengono svolte solo se lo smartphone non è connesso ad internet

5.4.1 Aggiungere un nuovo beacon

Questo sequence diagram mostra come viene aggiunto un nuovo beacon.
Osservazioni:

- Si ipotizza che i permessi per il bluetooth siano stati già dati all'applicazione e che il GPS sia attivo

- Se, dopo aver scaricato la lista dei device compatibili, l'applicazione non fosse più connessa ad internet l'evento di *init* viene salvato in database locale e più tardi si riproverà a mandare al server.

- Alla fine di tutte le operazioni l'utente verrà reindirizzato sulla pagina personale del beacon

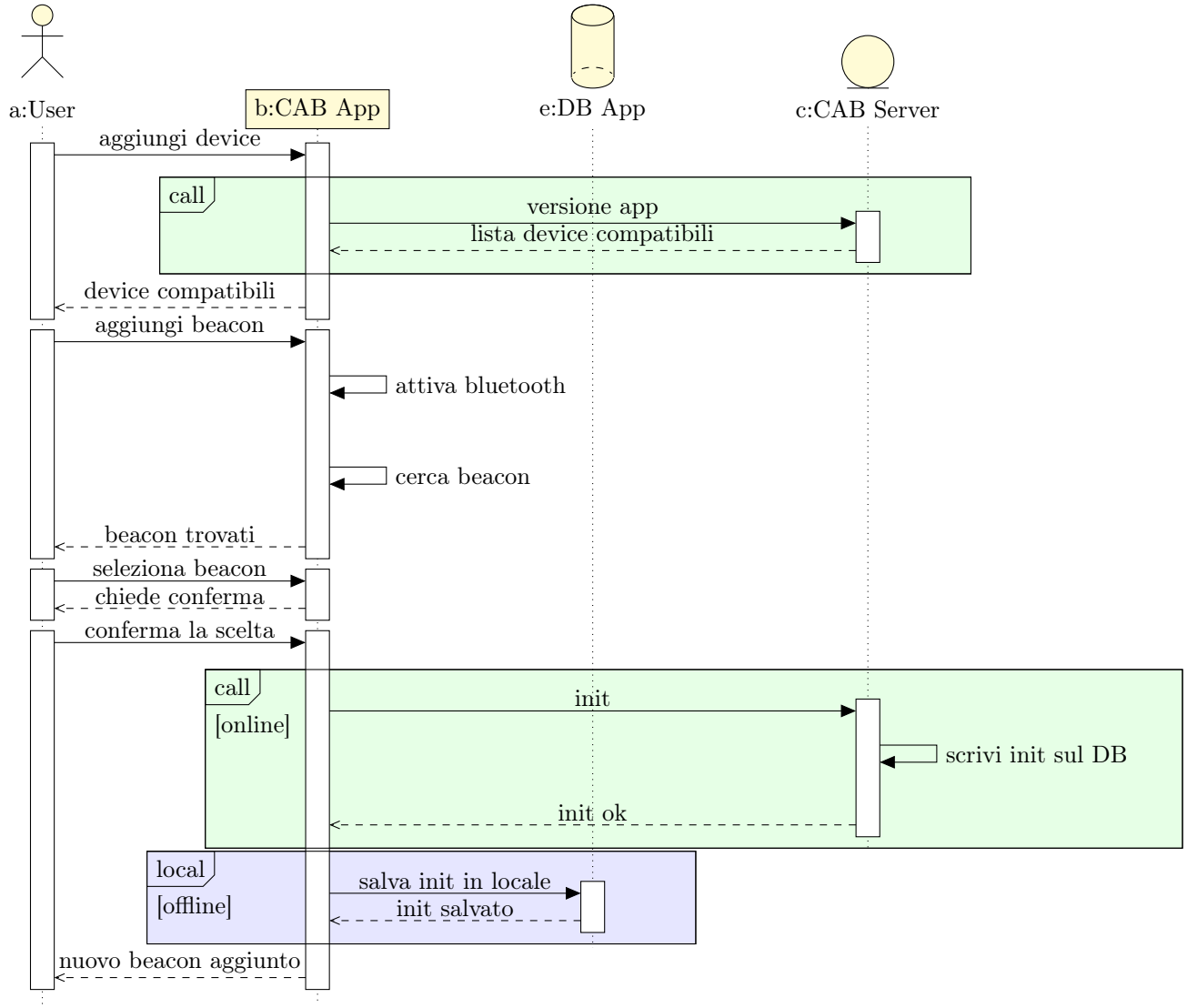


Figura 5.14: Seq.Diagram: Aggiungere un beacon

5.4.2 Aggiungere un nuovo robot da cucina

Questo sequence diagram mostra come viene aggiunto un nuovo robot da cucina.

Osservazioni:

- Si ipotizza che i permessi per il bluetooth siano stati già dati all'applicazione e che il GPS sia attivo
- Se, dopo aver scaricato la lista dei device compatibili, l'applicazione non fosse più connessa ad internet l'evento di *init* viene salvato in database locale e più tardi si riproverà a mandare al server
- Alla fine di tutte le operazioni l'utente verrà reindirizzato sulla pagina personale del robot da cucina

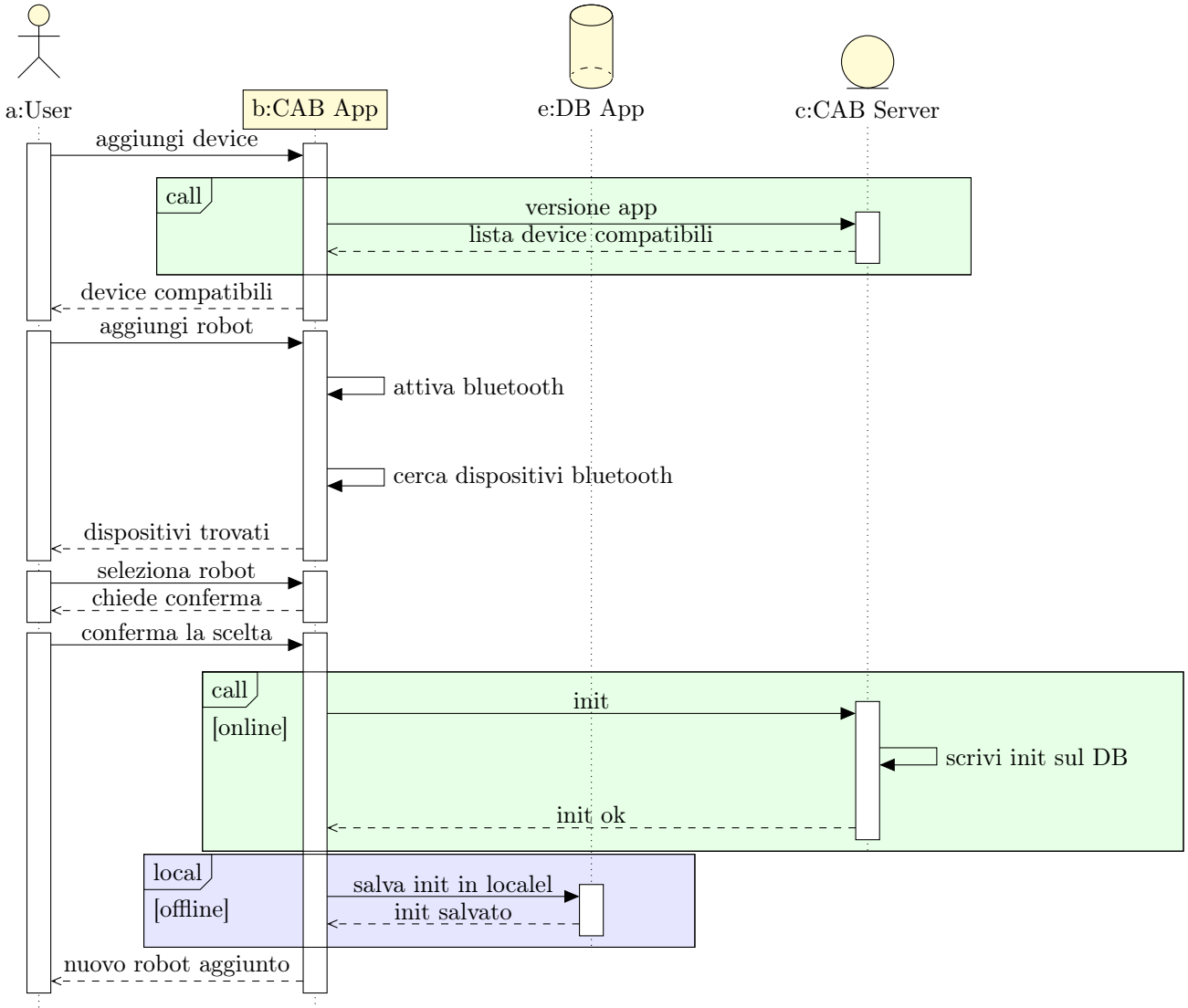


Figura 5.15: Seq.Diagram: Aggiungere un robot da cucina

5.4.3 Connettersi ad un beacon

Questo sequence diagram mostra come connettersi ad un beacon già aggiunto dall'utente.
Osservazioni:

- Finché l'utente non attiva il GPS non può ne connettersi al beacon ne accedere alla pagina di quest'ultimo
- L'utente viene indirizzato alla pagina del beacon anche se l'applicazione non è riuscita a connettersi ad esso, semplicemente visualizzerà lo stato di *non trovato* che verrà aggiornato non appena verrà rilevato il beacon
- Si da per scontato che l'utente abbia già dato i permessi per gestire il bluetooth perché altrimenti non avrebbe nemmeno aggiunto il beacon

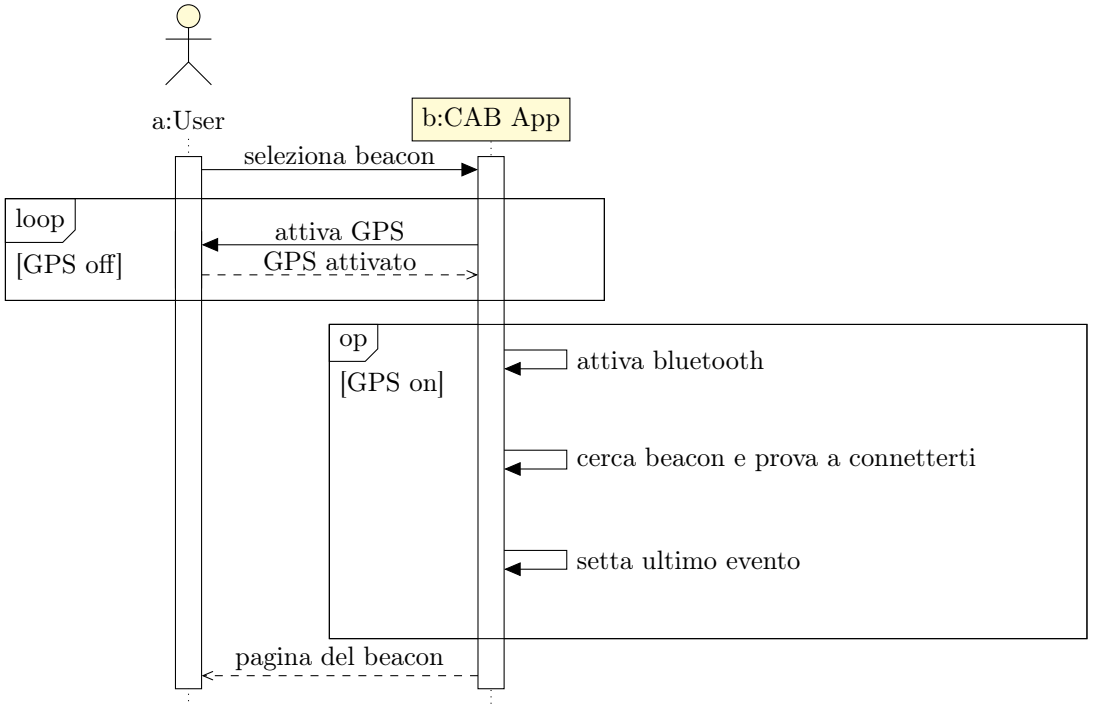


Figura 5.16: Seq.Diagram: Connettersi ad un beacon

5.4.4 Connettersi ad un robot da cucina

Questo sequence diagram mostra come connettersi ad un robot da cucina già aggiunto dall'utente.
Osservazioni:

- Finché l'utente non attiva il GPS non può ne connettersi al robot da cucina ne accedere alla pagina di quest'ultimo
- L'utente viene indirizzato alla pagina del robot da cucina anche se l'applicazione non è riuscita a connettersi ad esso, semplicemente visualizzerà lo stato di *non connesso* che verrà aggiornato non appena avverrà la connessione
- Si da per scontato che l'utente abbia già dato i permessi per gestire il bluetooth perché altrimenti non avrebbe nemmeno aggiunto il robot da cucina

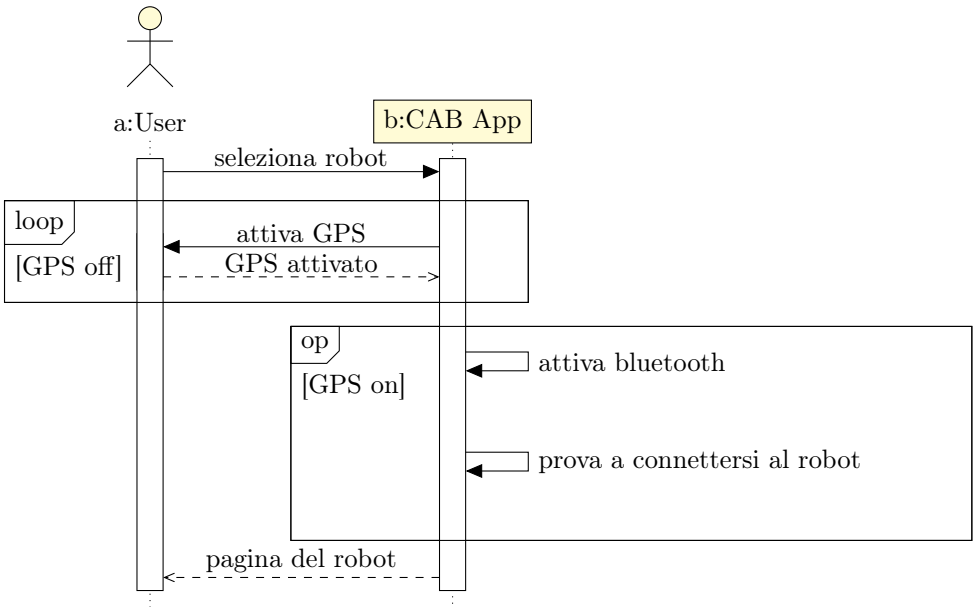


Figura 5.17: Seq.Diagram: Connettersi ad un robot da cucina

5.4.5 Gestione dell'invio degli eventi del beacon al server

Questo sequence diagram mostra come gli eventi generati dal beacon vengono inviati al server.
Gli eventi generati dal beacon sono:

- moving
- not moving
- lost
- found

Osservazioni:

- Il beacon viene perso se si allontana troppo dallo smartphone
- Nel sequence diagram per semplicità non viene mostrato l'utente che, ovviamente, sarà la causa degli eventi generati
- L'ultimo stato letto dal beacon viene salvato in una variabile di preferenza
- Gli eventi raccolti nel database locale vengono eliminati dopo essere stati inviati
- Più eventi vengono inviati al server attraverso un array di oggetti JSON
- In caso di errore di rete, ed evento non arrivato al server, il messaggio viene salvato in un database apposito dal quale si riproverà ad inviarlo successivamente (per i dettagli andare al paragrafo 5.4.7)

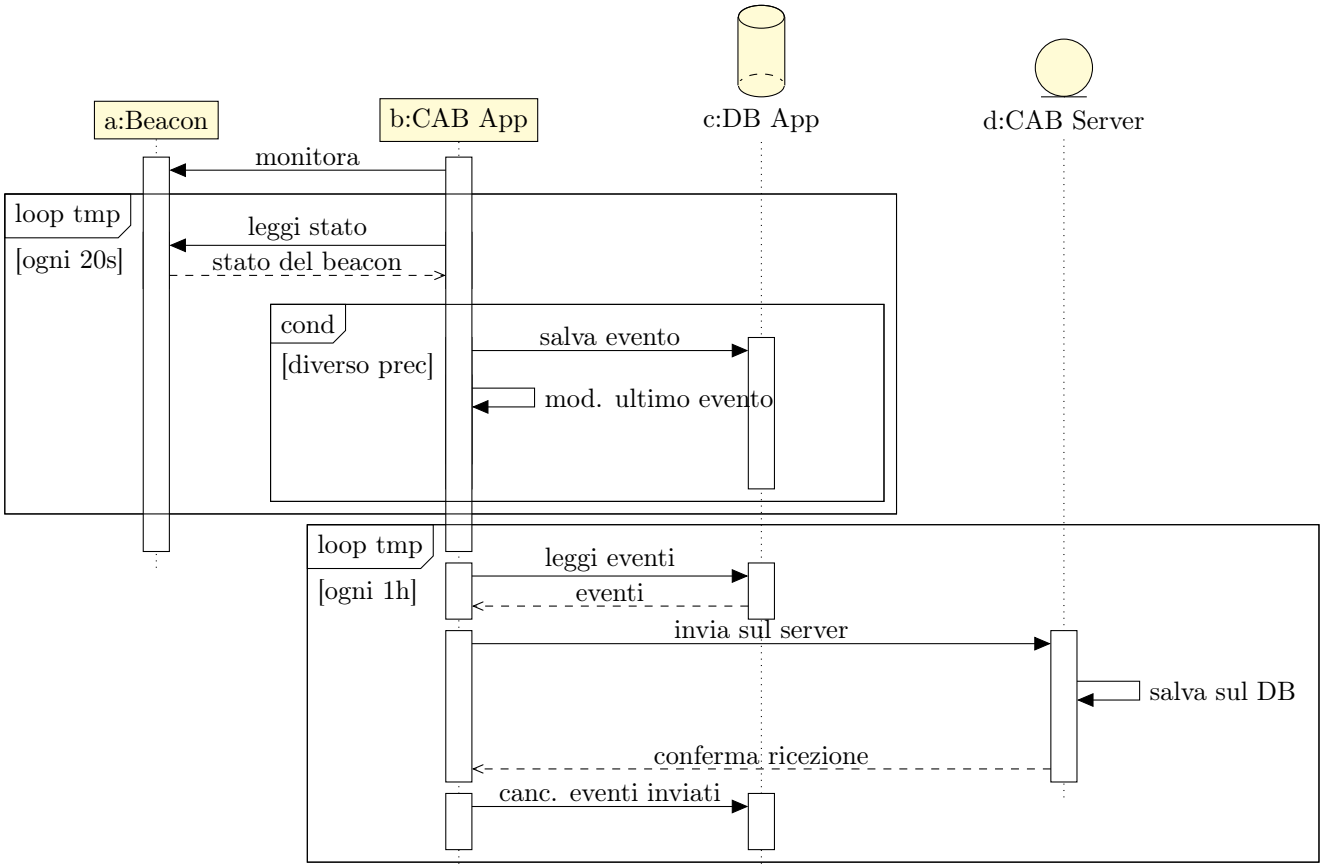


Figura 5.18: Seq.Diagram: Invio eventi generati dal beacon al server

5.4.6 Gestione dell'invio degli eventi del robot da cucina al server

Questo sequence diagram mostra come gli eventi generati dal robot da cucina vengono inviati al server.

- In caso di errore di rete, ed evento non arrivato al server, il messaggio viene salvato in un database apposito dal quale si riproverà ad inviarlo successivamente (per i dettagli andare al paragrafo 5.4.7)

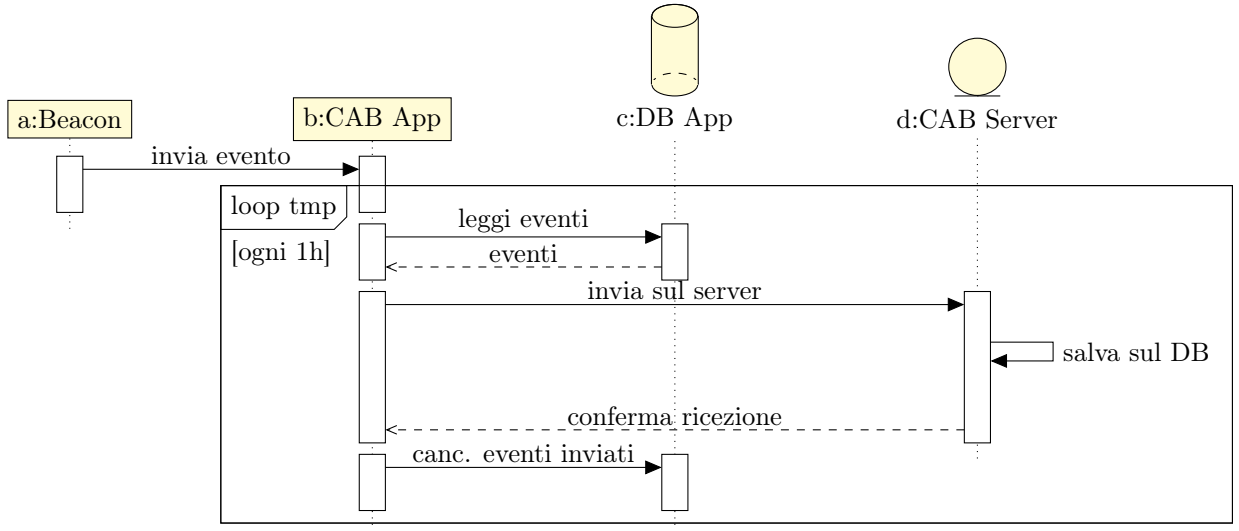


Figura 5.19: Seq.Diagram: Invio eventi generati dal robot da cucina al server

5.4.7 Gestione degli errori di rete per l'invio degli eventi

Questo sequence diagram mostra come, in caso di errore di rete, venga gestito il rinvio degli eventi al server.

Quando un messaggio, contenente un evento, non raggiunge il server a causa di un errore di rete il modulo all'interno dell'applicazione che si occupa della connessione salva in una tabella il *JSON* con il respettivo *URL*. Quando si vorrà provare il rinvio basterà leggere ciascuna riga della tabella e riprovare l'invio chiamando il metodo finale dato che *JSON e URL* vengono salvati nel formato necessario per essere inviati.

Osservazioni:

- Per ciascuna riga nella tabella vi è anche un campo che conteggia il numero di tentativi di rinvio del JSON presente nella riga stessa
- Se durante il rinvio si verifica di nuovo un errore di rete viene incrementato il campo che conteggia il numero di rinvii e il JSON, con il rispettivo URL, non viene ancora eliminato dalla tabella
- Nella tabella ciascuna riga viene eliminata se il rispettivo JSON è stato mandato con successo al server oppure se il numero di tentativi per inviarlo ha superato un certo numero

- Dopo un certo numero di tentativi (ad esempio 5) viene eliminato il JSON dal database locale per non rischiare che diventi grande occupando troppa memoria sullo smartphone a causa di numerosi errori di rete

- Un JSON può contenere un singolo evento ma anche un array di eventi (nel caso di un beacon)

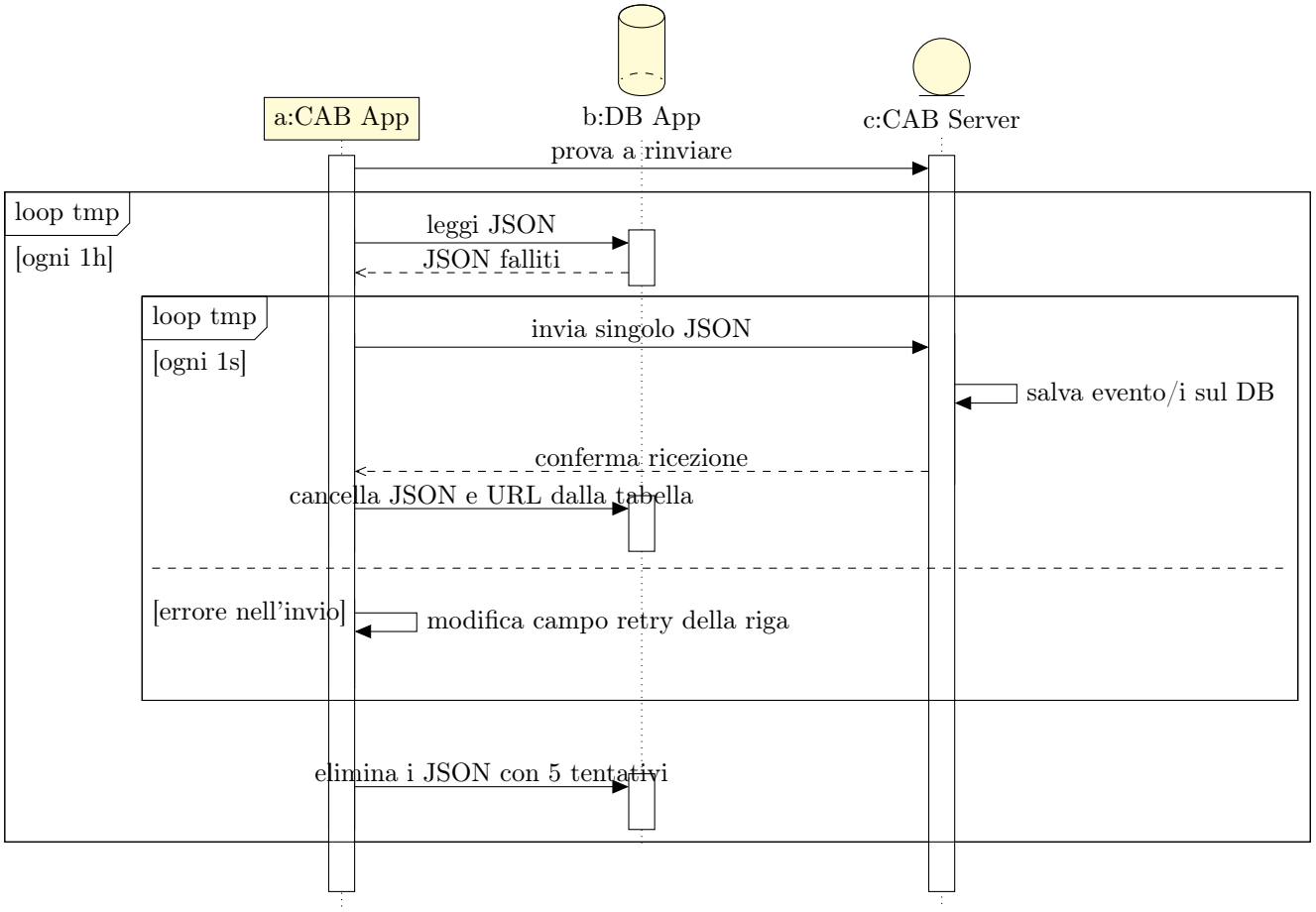


Figura 5.20: Seq.Diagram: Gestione degli errori di rete per l'invio degli eventi

5.5 Class Diagram

In questa sezione si presenteranno le principali classe (tutte sarebbero troppe) implementate nell'applicazione e come sono collegate tra di loro.

Osservazioni valide per tutti i grafici

- Per rendere il grafico più leggibile si è usato l'abbreviativo *Frag* per fragment e *Acti* per activity

- in tutti i class diagram fatti non sono presenti tutte le classi che costituiscono l'applicazione in un caso e la libraria negli altri; si è scelto di mettere al loro interno soltanto le principali classi per non rendere il class diagram illeggibile e quindi inutile; per questo motivo ciascuno di essi è accompagnato da un'ampia spiegazione.

5.5.1 Applicazione

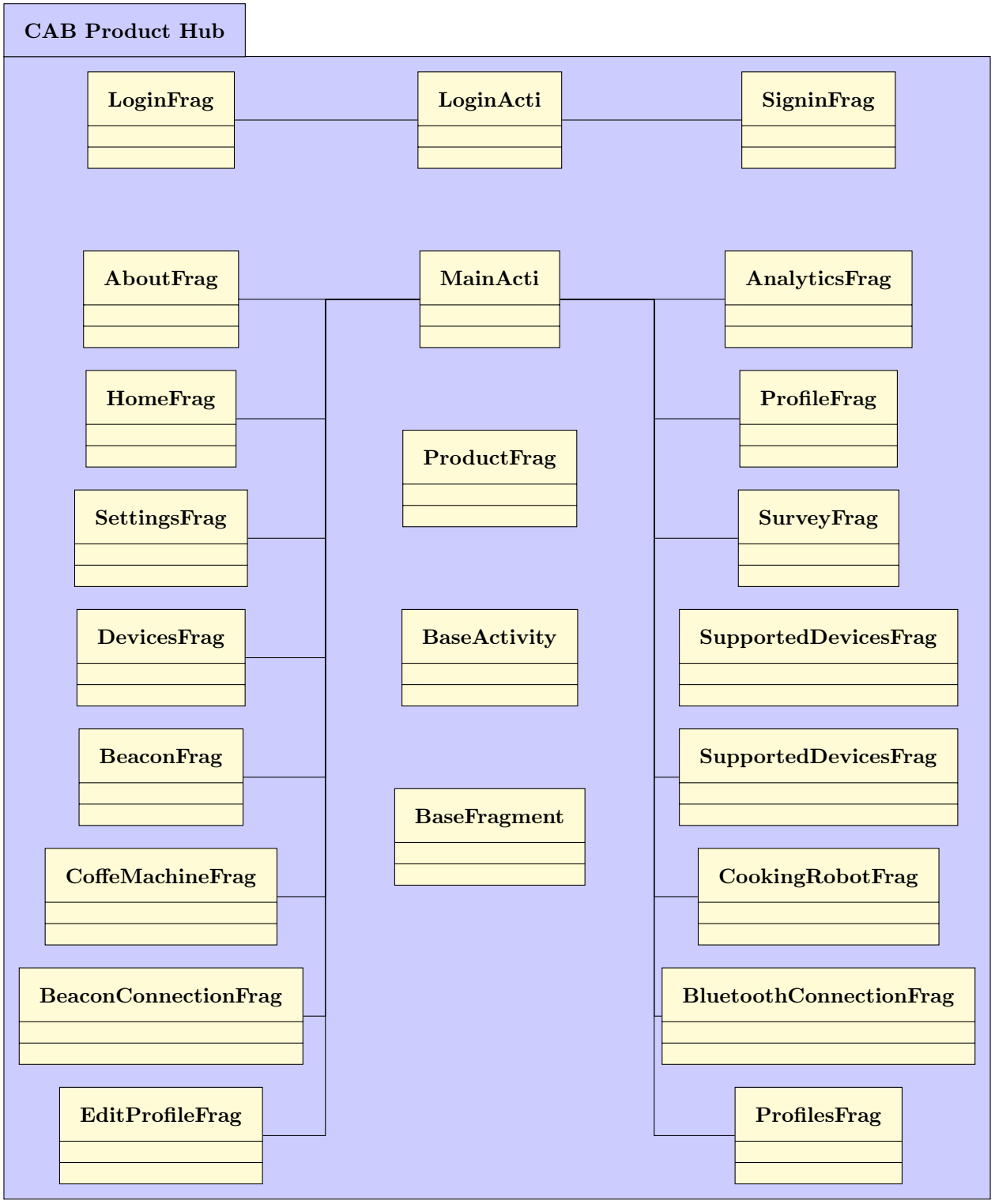


Figura 5.21: Class Diagram: Applicazione CAB Product Hub

Si nota subito che il grafico non un vero e proprio class diagram in quanto sono stati trascurati i collegamenti che riguardano le ereditarietà (per questo dalle 3 classi al centro non parte alcun collegamento) ed è stata fatta questa scelta per non rendere il grafico illeggibile a causa dell'elevato numero di connessioni che sarebbero state necessarie.
Nel grafico sopra sono evidenziate le principali classi che costituiscono il progetto e come si passa da una all'altra durante la navigazione da parte dell'utente, e cioè attraverso dei metodi creati nella'activity principale.
Sono presenti due activity, una che gestisce il login e la registrazione e una che si occupa di tutti i fragment e servizi presenti nell'applicazione. Si hanno poi diversi fragment, uno per ogni pagina presente nell'applicazione.
Tutti i fragment e le activity estendono le classi presenti al centro dello schema, *BaseActivity* e *BaseFragment*. L'unica eccezione è rappresentata dai tre fragment dedicati ai dispositivi (*BeaconFrag, CoffeMachineFrag e CookingRobotFrag*) che estendono *ProductFrag*, un fragment contenente metodi comuni a tutti i dispositivi, come quelli che gestiscono la rimozione del dispositivo e i permessi di inviare informazioni. Le classi che vengono estese contengono metodi che è necessario avere in tutte le classi che le estendono e così facendo si evita di riscrivere lo stesso codice su più classi.
Tra le classi non presenti nello schema, che altrimenti sarebbe diventato illeggibile, si hanno tra le più importanti:

- *CabManager* che gestisce tutte le chiamate al server riguardanti la lettura delle novità, delle liste dei prodotti che l'utente ha già aggiunto o può aggiungere, e i metodi per aggiungere e rimuovere i prodotti dalla lista dei prodotti aggiunti dall'utente e salvata sul server online

- *CabConstants* contiene alcune importanti costanti globali utilizzate in varie parti del codice

- *CheckInternetConnection* controlla lo stato connessione attraverso un listener che si attiva ogni volta che esso cambia

- *SharedPreferenceDelegate* gestisce le varie preferenze salvate nell'applicazione permettendo di settarle o leggerle

- *VolleyService* Si occupa di inizializzare il servizio di Volley usato poi nelle librerie dei beacon e del robot da cucina per inviare gli eventi al server
- *VolleySingleton* utilizzato per caricare, attraverso l'url, le immagini salvate online in maniera intelligente sfruttando meccanismi di cache

5.5.2 Libreria beacon

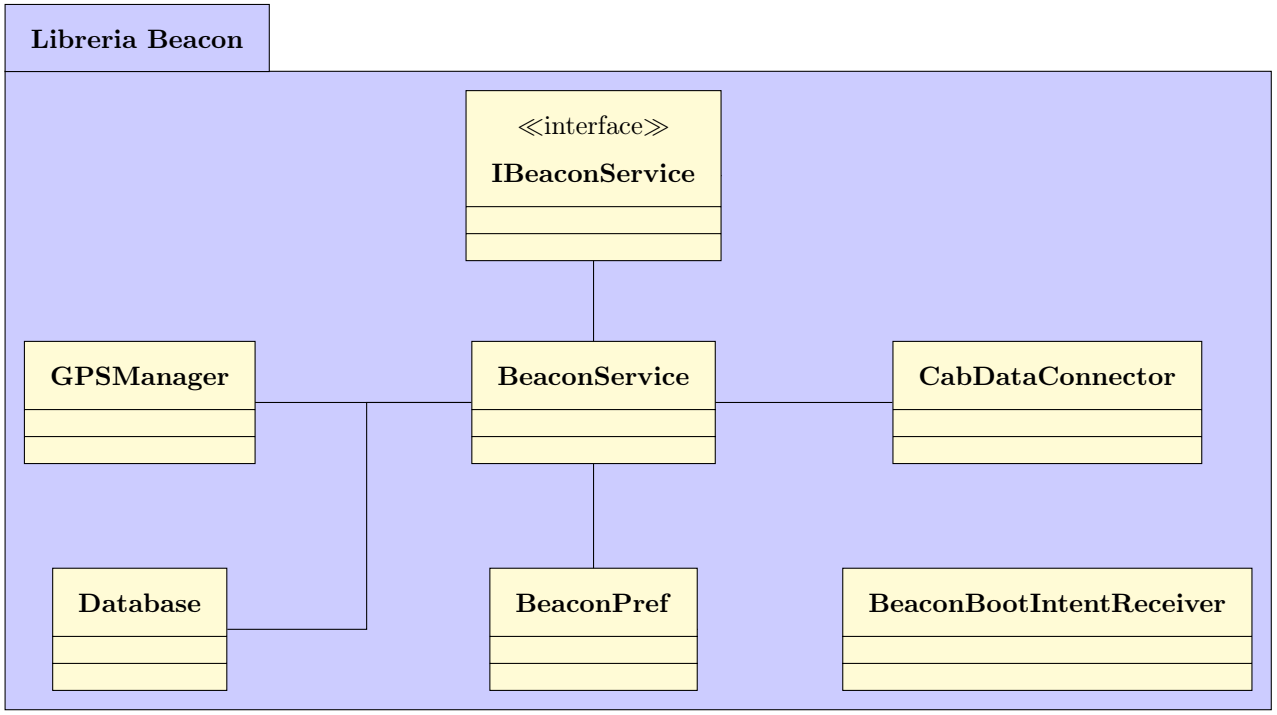


Figura 5.22: Class Diagram: Libreria beacon

La classe intorno alla quale ruota tutta la libreria è, come si può ben intuire anche dal grafico, *BeaconService*. Essa implementa un'interfaccia che viene usata poi dall'applicazione in cui si intende importare la libreria. Tale pattern è necessario a separare i servizi che offre la libreria dal come li offre nascondendo così l'implementazione della classe agli occhi dello sviluppatore che la usa.
Inoltre estende la classe astratta *Service()* che permette al servizio offerto da BeaconService, e cioè l'invio degli eventi generati dal beacon, di continuare a funzionare anche se l'utente non ha l'applicazione sempre aperta. Le altre classi usate da BeaconService sono:

- *GPSManager* che gestisce il servizio che ritorna la posizione usando le API di Google

- *BeaconPref* gestisce le preferenze salvate dalla libreria permettendo di settarle o leggerle
- *CabDataConnector* gestisce l'invio degli eventi generati dal beacon al server
- *Database* gestisce il database in cui vengono salvati gli eventi generati dai beacon prima di essere mandati al server

Infine vi è la classe *BeaconBootIntentReceiver* che si occupa di far ripartire il servizio di BeaconService quando lo smartphone, spento, viene riacceso.

5.5.3 Libreria connessione

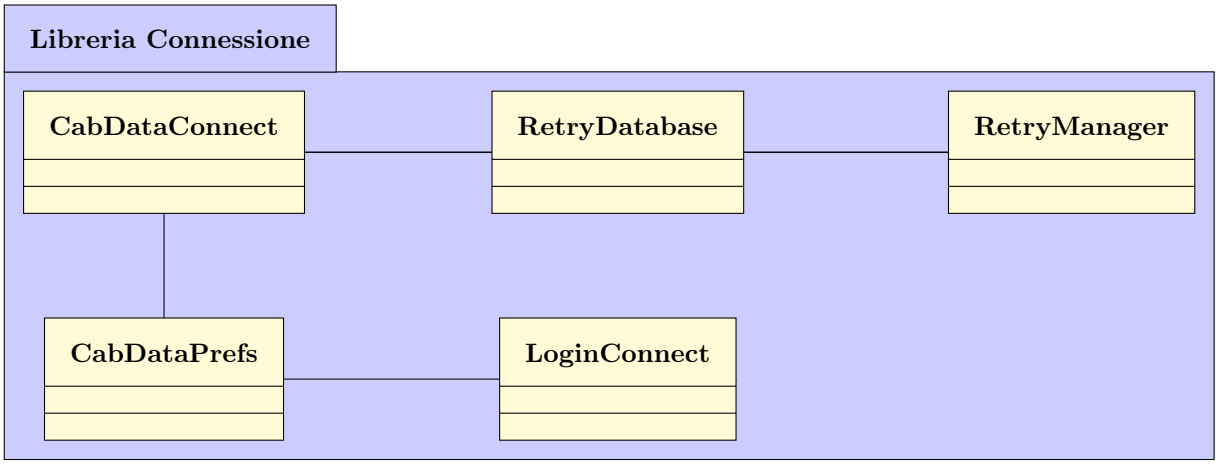


Figura 5.23: Class Diagram: Libreria connessione

La maggior parte di classi che costituiscono questa libreria, non presenti nel grafico, è costituita da classi definite *Serilizable* usate come modelli in cui salvare i dati per poi trasformarli in file JSON, usando l'apposita libreria Gson, da inviare nelle richieste al server. Inoltre, al contrario, è possibile trasformare un JSON ricevuto dal server in una di queste classi che conterrà i dati ricevuti.
Le altre classi presenti nella libreria si occupano di gestire le comunicazioni con il server:

- *CabDataConnect* è la classe più importante, gestisce tutte le richieste POST che, nell'applicazione che la usa, mandano al server i dati generati dai vari prodotti
- *RetryDatabase* è il database che contiene tutte le richieste fallite durante l'invio al server

- *RetryManager* si occupa, usando CabDataConnect, di riprovare, periodicamente, ad inviare tutte le richieste fallite contenute nel RetryDatabase; inoltre se le richieste falliscono per un certo numero di tentativi, si è utilizzato 5 come limite massimo, tali richieste vengono eliminate per non occupare troppa memoria sul dispositivo

- *LoginConnect* gestisce le chiamate fatte al server per l'accesso e la registrazione

- *CabDataPrefs* gestisce tutte le preferenze salvate nella libreria permettendo di settarle o leggerle

5.6 Interfaccia Utente

In questo capitolo si mostreranno i screenshot delle principali schermate presenti all'interno dell'applicazione con una breve descrizione per ognuna di esse.

5.6.1 Login e Registrazione

In queste due pagine l'utente può effettuare il login e la registrazione.
La pagina della registrazione era raggiungibile da quella di login ma successivamente è stata rimossa perché non sono state fornite le API necessarie per effettuarla. Al posto della pagina l'utente viene reindirizzato nel sito del produttore, dove può effettuare la registrazione. Tuttavia la pagina e i metodi sono ancora presenti nel codice, anche se non visibili, e pronti a funzionare non appena si avranno le API a disposizione.

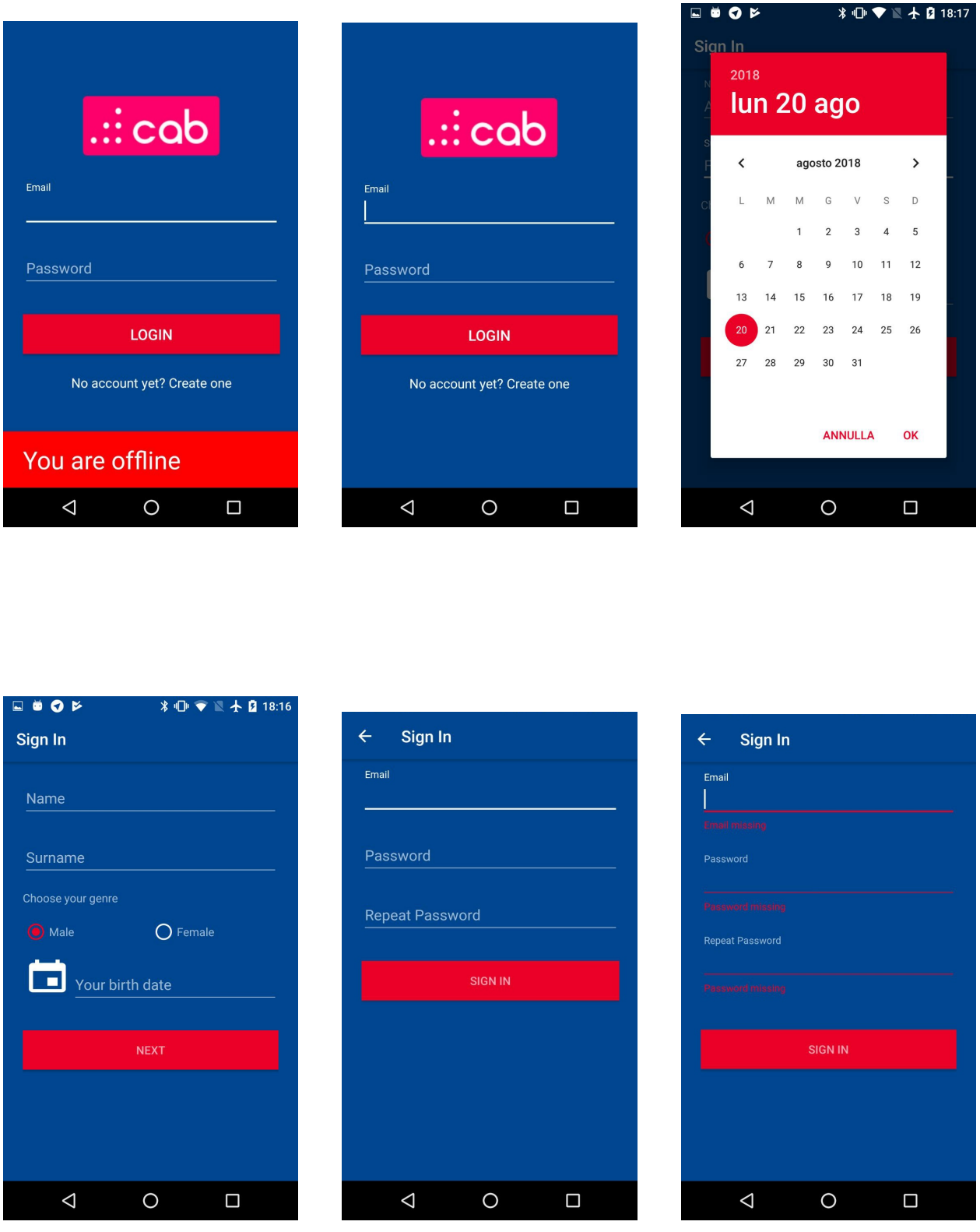


Figura 5.24: Interfaccia Utente: Login e registrazione

5.6.2 Home

Questa è la prima pagina in cui l'utente viene indirizzato una volta effettuato il login.
Vi era l'idea di mostrare una lista con le novità relative ai prodotti aggiunti dall'utente. Tuttavia tale funzionalità non è stata implementata nel corso di questo progetto ma è presente una lista di notizie casuali a scopo dimostrativo.

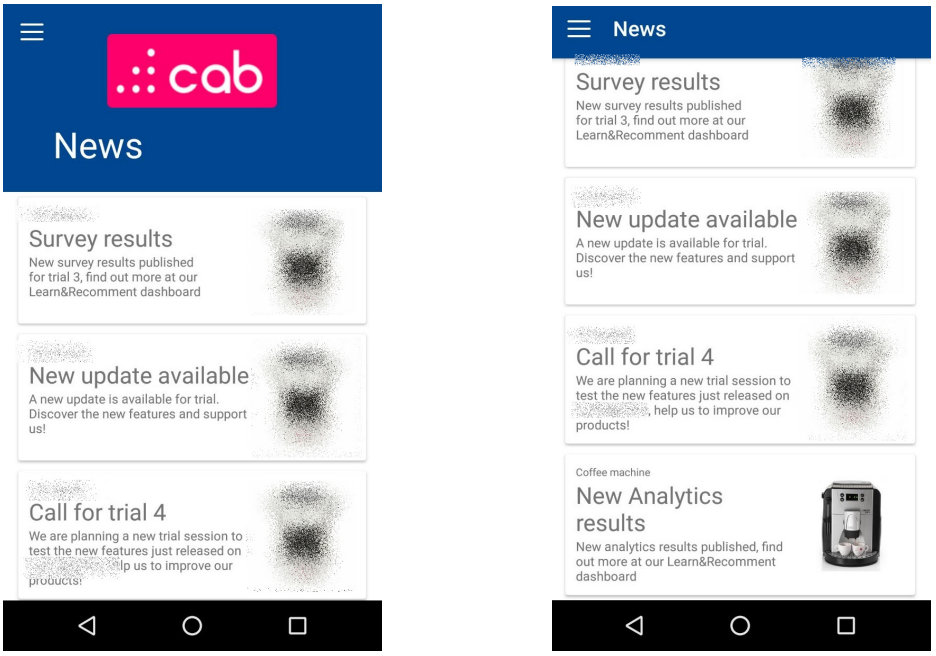


Figura 5.25: Interfaccia Utente: Home

5.6.3 Menù a comparsa

Il menù a comparsa da sinistra, praticamente uno standard in moltissime applicazioni, permette all'utente di muoversi tra le diverse pagine dell'applicazione.
Nell'icona in alto a sinistra si possono vedere il nome e la foto del profilo attualmente attivo per raccogliere i dati. Inoltre è visibile il nome dell'account.

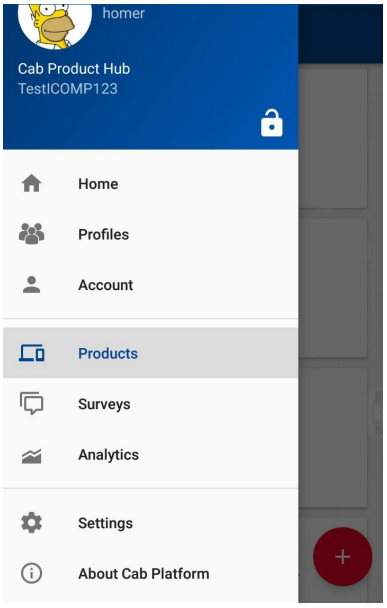


Figura 5.26: Interfaccia Utente: Menu laterale

5.6.4 Pagina dei dispositivi

In questa pagina l'utente può vedere tutti i dispositivi da lui aggiunti e, tramite il pulsante rosso in basso a destra, aggiungerne di nuovi andando alla pagina che contiene la lista completa di tutti i dispositivi supportati dall'applicazione.

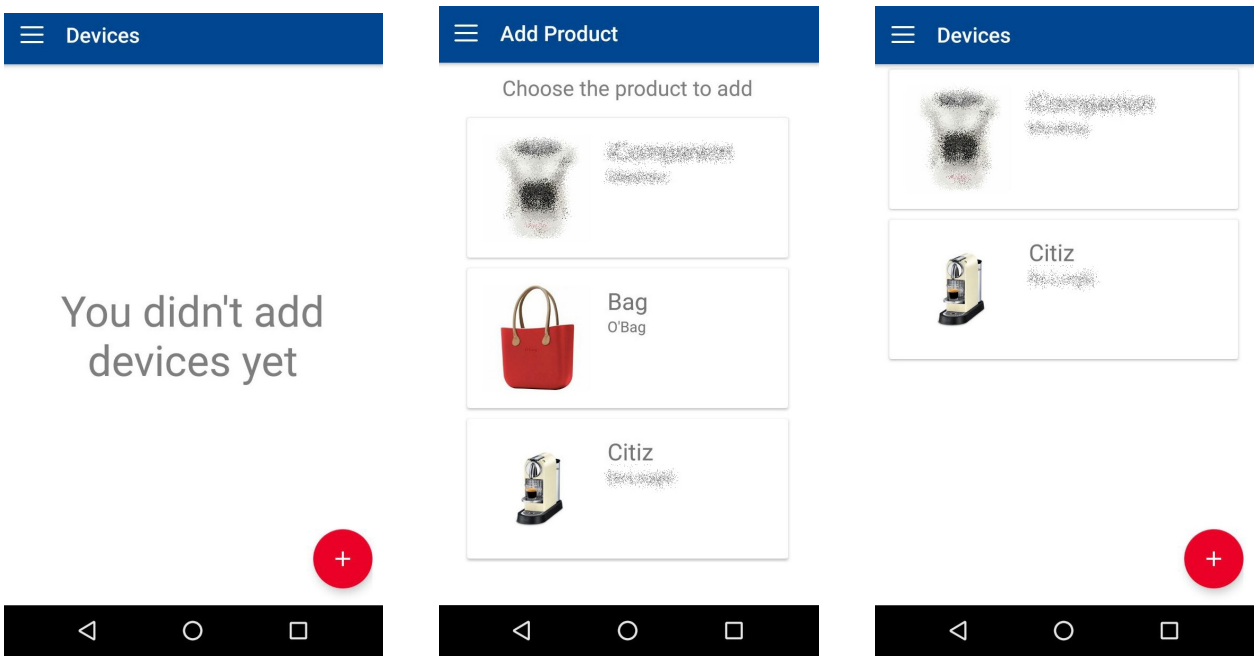


Figura 5.27: Interfaccia Utente: Pagine dei dispositivi

5.6.5 Beacon

Se si vuole aggiungere un nuovo beacon è possibile selezionarlo dalla pagina del dispositivi supportati. Da lì si andrà alla lista dei beacon rilevati dall'applicazione dove l'utente potrà selezionare il proprio beacon distinguibile dal nome dell'immagine impressa sopra il beacon stesso.
Se invece il beacon è già stato aggiunto si può raggiungere la sua pagina dedicata dalla lista iniziale dei dispositivi già aggiunti.
In tale pagina l'utente potrà vedere lo stato del beacon e gestirne i permessi scegliendo se mandare ogni tipo di dato, soltanto i dati relativi l'utilizzo tralasciando la posizione o non inviare nessun tipo di dato.

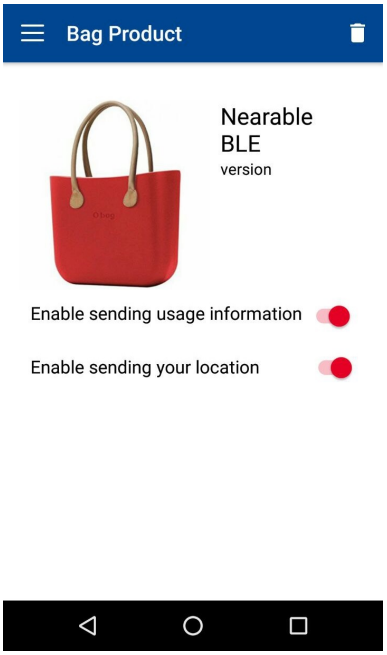


Figura 5.28: Interfaccia Utente: Beacon

5.6.6 Robot da cucina

Se si vuole aggiungere un nuovo robot da cucina è possibile selezionarlo dalla pagina del dispositivi supportati. Da lì si andrà alla lista dei dispositivi rilevati dall'applicazione dove l'utente potrà selezionare il proprio robot da cucina, distinguibile dal nome.
Se invece il robot da cucina è già stato aggiunto si può raggiungere la sua pagina dedicata dalla lista iniziale dei dispositivi già aggiunti.
In tale pagina l'utente potrà vedere l'ultima funzione attivata sul robot, il suo stato di connessione con l'applicazione e gestirne i permessi scegliendo se mandare o meno i dati.

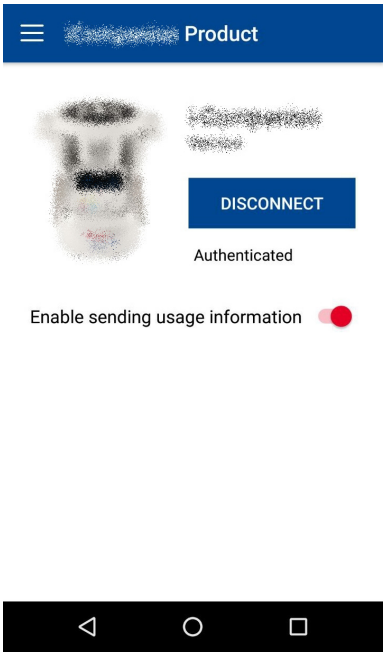


Figura 5.29: Interfaccia Utente: Robot da cucina

5.6.7 Macchina del caffè

Se si vuole aggiungere una nuova macchina del caffè è possibile selezionarlo dalla pagina del dispositivi supportati.
Se invece la macchina è già stata aggiunta si può raggiungere la sua pagina dedicata dalla lista iniziale dei dispositivi già aggiunti.
Dato che non avviene una connessione diretta tra la macchina del caffè e lo smartphone in questa pagina l'utente può soltanto gestire i permessi sull'invio dei dati.

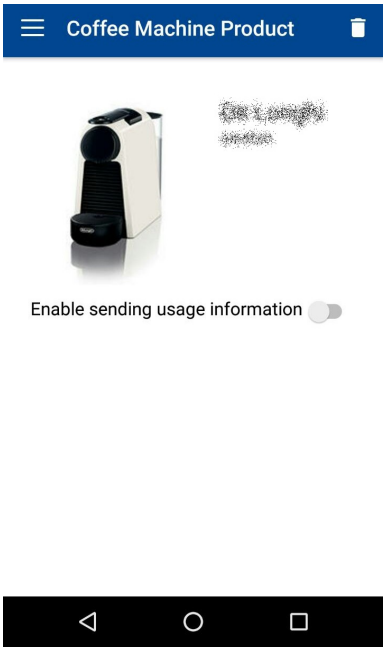


Figura 5.30: Interfaccia Utente: Macchina del caffè

5.6.8 Impostazioni

La pagina delle impostazioni al momento fornisce soltanto la versione dell'applicazione e, tramite due tasti rapidi sulla versione stessa, permette di aprire un menù per gli sviluppatori.

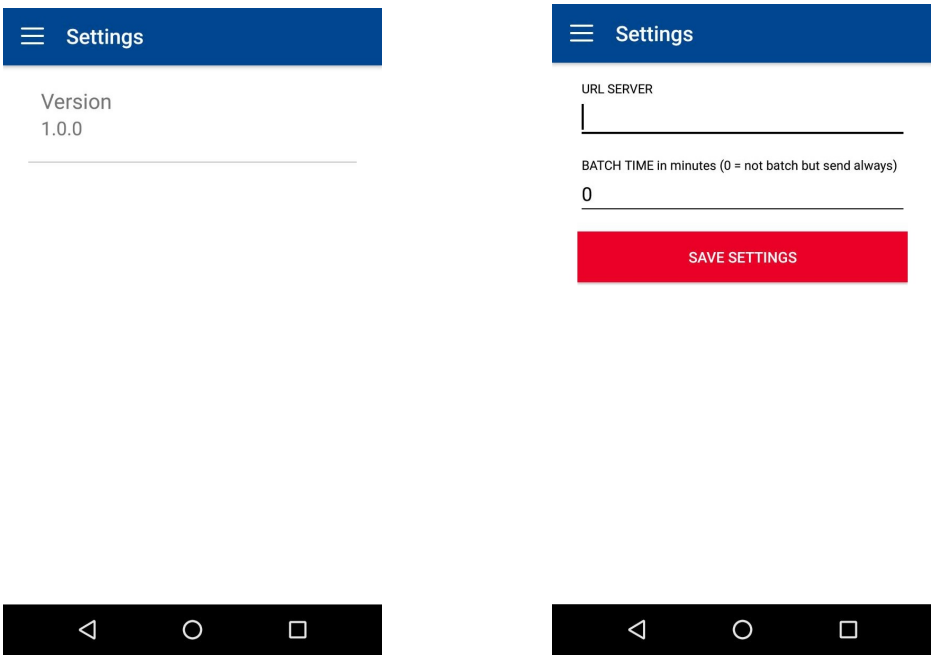


Figura 5.31: Interfaccia Utente: Impostazioni

5.6.9 Account

In questa pagina è visualizzabile il nome dell'account e da qui sarà possibile eliminare il proprio account. Tale funzione non è stata implementata durante il progetto.

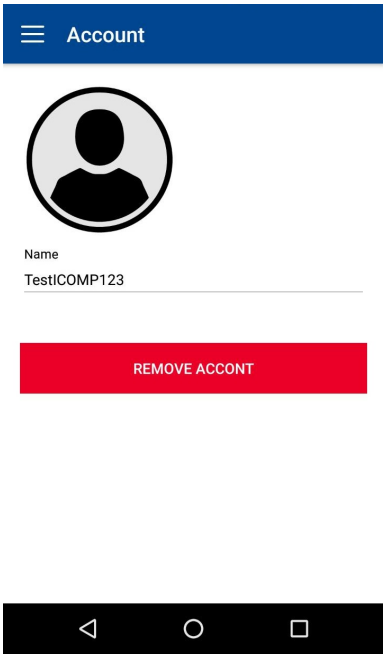


Figura 5.32: Interfaccia Utente: Account

5.6.10 Gestione profili

Tramite queste due pagine l'utente può gestire i vari profili aggiungendone di nuovi o eliminandone qualcuno. E, per ciascun profilo, può gestirne gli "identifier", cioè gli identificatori che permettono di cambiare il profilo attivo rapidamente.
L'unica tipologia di identificatore sviluppata durante la tesi è rappresentata dai badge che utilizzano tecnologia NFC.

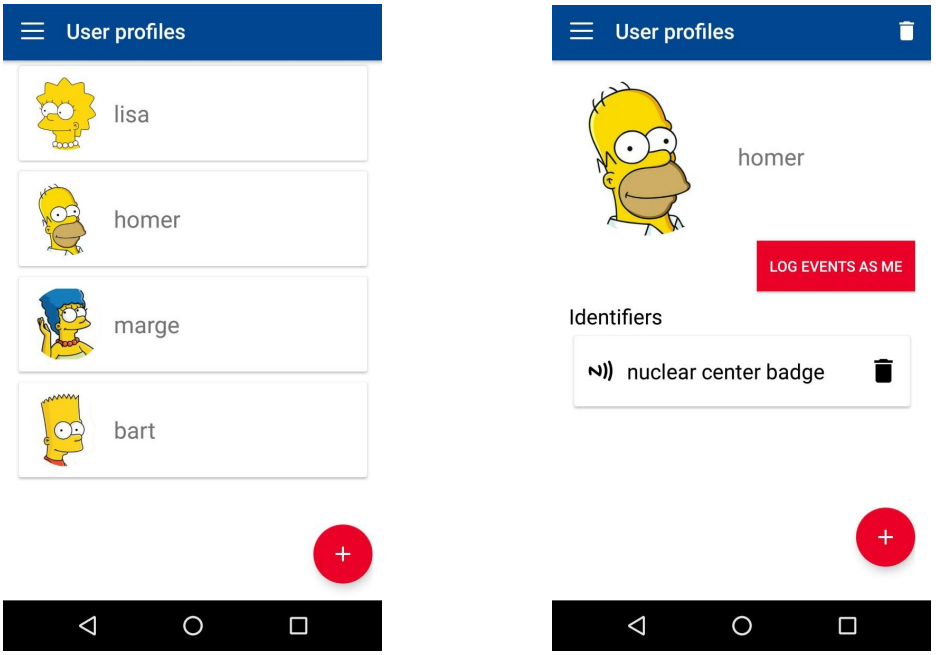


Figura 5.33: Interfaccia Utente: Profili

5.6.11 About

Tramite una *web view* visualizza alcune informazioni riguardo l'applicazione e l'intero progetto.

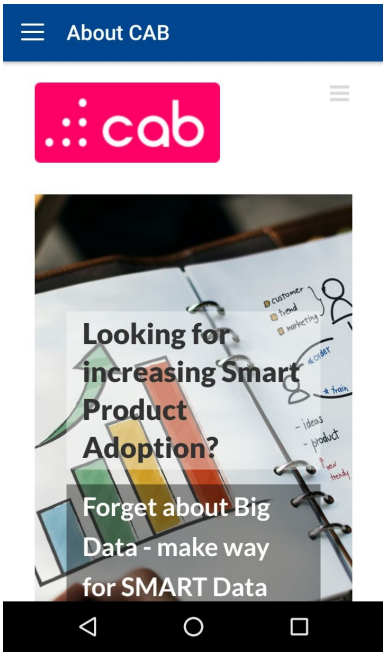


Figura 5.34: Interfaccia Utente: About

5.7 Attributi del Sistema

5.7.1 Affidabilità

I dati raccolti dal sistema vengono salvati in due diversi server, uno dei quali viene utilizzato anche per esperimenti in fase di sviluppo.

5.7.2 Disponibilità

Per accedere all'applicazione, e usufruire dei vari servizi offerti da essa, è necessario avere a disposizione una connessione internet.
Se però si perde la connessione internet mentre l'applicazione è connessa ad un dispositivo di cui si stanno raccogliendo i dati, tali dati non vengono persi ma, una volta fallito il loro invio al server, vengono salvati in un database locale e si riproverà ad inviarli ad intervalli di tempo regolari (durante i test si è optato per un tentativo ogni cinque ore).

5.7.3 Sicurezza

Per poter scambiare dati con i server è necessario effettuare delle chiamate *POST* con inserimento nell'*header* del campo di autenticazione con il codice assegnato all'utente durante la creazione di una sessione valida una volta effettuato il login.
Non si è quindi in grado di comunicare con il server se non ci si è autenticati ed inoltre, usando *POST*, i dati non sono passati in chiaro.

5.7.4 Manutenibilità

Si è sviluppata l'applicazione suddividendola in vari moduli e creando anche librerie completamente separate e indipendenti tra loro. In questo modo l'applicazione si può aggiornare facilmente aggiungendo nuove funzionalità o modificando, al fine di perfezionarle, adattarle o correggerle, quelle esistenti.

5.7.5 Usabilità

Si è cercato di creare un'applicazione più *user friendly* possibile optando per scelte grafiche standard e semplici utilizzate in varie applicazioni usate da miliardi di utenti come il menu a comparsa o la posizione di alcuni bottoni all'interno dell'applicazione. Tutto questo per permettere all'utente di capire immediatamente come funzionano i vari servizi offerti.
Inoltre altre scelte grafiche sono state inserite per rendere l'applicazione più accattivante in modo da invogliare, per quanto un'applicazione di questo tipo possa fare, l'utente ad utilizzarla.

5.7.6 Portabilità

L'applicazione è stata sviluppata in Kotlin e Java sull'ambiente di sviluppo ufficiale Android Studio. Pertanto l'applicazione può essere utilizzata soltanto in dispositivi con sistema operativo Android.

5.8 Principali Librerie Usate

In questo capitolo si descriveranno le principali librerie usate nell'applicazione, eventuali alternative, e le motivazioni per le quali sono state usate.

5.8.1 Realm

Realm è un sistema di gestione per database open source sviluppato per applicazioni mobile e, insieme a SQLite, rappresenta la libreria più utilizzata in tale campo.
Essendo Realm un prodotto nuovo, almeno rispetto a SQLite, presenta diverse migliorie senza contare che Realm è nato appositamente per gestire database su applicazioni mobile mentre SQLite è un gestore di database per praticamente qualsiasi piattaforma.
I principali vantaggi per quale si è scelto di usare Real sono:

- **Semplicità** Non è un database relazionale nel quale bisogna creare tabelle, ma potremmo definirlo un *"database ad oggetti"* perché al posto delle tabelle abbiamo delle classi i cui attributi solo l'equivalente delle colonne in una tabella; questo comporta che con pochissime righe di codice si possono inserire nuovi dati, modificarne o estrarne (con una sola riga si possono estrarre i dati settando pure delle condizioni) velocizzando anche la programmazione e minimizzando gli errori
- **Velocità** in molti casi le stesse operazioni sono risultate molto più veloci su Realm che su SQLite il che lo rende anche più scalabile[14]
- **Accessibilità** permette di copiare i suoi oggetti che rimangono accessibili anche dall'esterno
- **Caratteristiche aggiuntive** essendo più giovane presenta caratteristiche bonus che possono aiutare il programmatore come il supporto per JSON e per la criptazione dei dati

Per concludere, in questo progetto si è scelto Realm sopratutto per la sua semplicità, dato che si ha un database dalla struttura molto semplice, e per la sua scalabilità, dato che il numero di dati potrà crescere in maniera esponenziale con il crescere di prodotti supportati e utilizzatori.[15]
Vediamo sotto un semplice esempio di un oggetto Realm e di come viene utilizzato per un inserimento asincrono per far meglio comprendere la definizione di *"database ad oggetti"* detta sopra.

```
open class MyModel : RealmObject() {
    private var Parametro_1: String? = null
    private var Parametro_2: String? = null
    fun setParametro_1(p: String) {
```

```
        Parametro_1 = p
    }
    fun getParametro_1(): String? {
        return Parametro_1
    }
    fun setParametro_2(p: String) {
        Parametro_2 = p
    }
    fun getParametro_2(): String? {
        return Parametro_2
    }
}
----------In un altra classe----------
fun asyncronousInsertNewEvent(P1: String, P2: String) {
        if (realm != null) {
            realm.executeTransactionAsync(Realm.Transaction { realm ->
                var model: MyModel = realm!!
                .createObject(MyModel::class.java)
                event.setParametro_1(p1)
                event.setParametro_2(p2)
            }, object : Realm.Transaction.OnSuccess {
                override fun onSuccess() {
                    Log.d(TAG, "Inserimento avvenuto")
                }
            }, object : Realm.Transaction.OnError {
                override fun onError(error: Throwable?) {
                    Log.d(TAG, "Inserimento fallito")
                }

            })
```

```
        }
    }
```

Nell'applicazione tutte le chiamate al server avvengono attraverso processi asincroni per non rallentare l'applicazione.

5.8.2 Volley

Volley è una libreria HTTP che si usa per gestire l'accesso alla rete delle applicazioni Android in maniera semplice e veloce.
In genere accedere a vari servizi di rete (come ad esempio effettuare una POST con JSON) è un'operazione condizionata da tempi di latenza e perciò dev'essere svolta in maniera asincrona per non rallentare l'intera applicazione, senza contare la gestione degli errori che si possono verificare. Vista l'importanza ti tali operazioni è stata predisposta proprio la libreria Volley che è in grado di curare tutti gli aspetti di un accesso alla Rete:

- Gestione autonoma delle richieste e connessioni multiple
- *Caching* delle risposte sia in memoria che su disco
- Varie classi per il supporto dei tipi più comuni di richieste (ad esempio JSONObjectRequest o JSONArrayRequest sono quelle usate nell'applicazione)
- Gestione delle priorità
- Strumenti di log, debug e tracciamento delle attività che permettono di monitorare le richieste inviate

Oltre ai vantaggi elencati sopra si è scelto Volley perché permette in maniera semplice, e con poche righe di codice, di fare delle POST con autenticazione e un JSON, o array di JSON, come corpo. Inoltre Volley gestisce il rinvio in caso di errore settando il numero di tentativi massimi che si vuole provare e il timeout massimo tra un tentativo è l'altro.[16]

5.8.3 Gson

Gson è una libreria open source originariamente sviluppata da Google per motivi interni utilizzata per serializzare e deserializzare un oggetto Java in/da JSON.[17]
Si è usata questa libreria per serializzare oggetti Java nei JSON che vengono poi mandati tramite un post al server. La si è riusata per deserializzare il JSON di ritorno dal server in oggetto Java.
Ci sono molte altre libreria che permettono di gestire i JSON, tuttavia si è scelta Gson semplicemente perché è stata sviluppata da Google e quindi si ha la sicurezza, più di qualsiasi altra libreria sviluppata da terze parti, che non avrà problemi di compatibilità di alcun tipo con l'applicazione.

5.8.4 Estimote

Estimote è la libreria ufficiale dell'omonima azienda che produce e vende i beacon.[22]
La libreria, periodicamente aggiornata, permette di gestire i beacon monitorando la loro presenza nei d'intorni, il livello di potenza del segnale che trasmettono (che dipenderà ovviamente dalla distanza) e altri parametri differenti in base al tipo di beacon. Esistono infatti vari beacon con all'interno sensori diversi in base all'utilizzo che se ne vuole fare.

Capitolo 6

Conclusioni

6.1 Valutazione della soluzione realizzata

Si è ampiamente soddisfatti del risultato finale raggiunto dal progetto seguito durante questa tesi.
Sia perché si è riusciti a realizzare un'applicazione per Android che potrebbe già essere usata in diversi casi d'uso reale che vedremo nella sezione successiva. Ma sopratutto perché è stato messo in piedi un sistema, connesso all'applicazione, che permette non solo di connettersi e raccogliere dati da diverse classi di prodotti ma anche di aggiungerne di nuovi facilmente grazie alla grande generalizzazione che è stata fatta durante il progetto.
Non ci si è concentrati soltanto sui tre prodotti utilizzati in pratica ma, sopratutto sviluppando l'applicazione, si è lavorato in ottica di una soluzione alla quale possono essere aggiunti diversi prodotti in maniera rapida e veloce, sfruttando parte del lavoro già fatto, senza dover partire da zero.

6.2 Casi d'uso reali

Il progetto è stato sviluppato scegliendo due particolari casi d'uso reale:

- *Domestico* in cui una famiglia riceve un prodotto gratuitamente da un'azienda in cambio di un periodo più o meno lungo di test; inoltre la stessa azienda manderà a casa della famiglia un

tecnico specializzato nel caso in cui ci sia qualcosa da installare, come ad esempio un piccolo server con OpenHAB nel caso di una classica macchina del caffè

- *Pubblico* in cui vengono monitorati prodotti utilizzati da ampi gruppi di persone come una macchina del caffè posto in un ufficio

Durante il progetto il beacon, inserito in una borsa, e la macchia del caffè sono stati gli unici dure prodotti testati in casi d'uso reale da alcuni membri del laboratorio che, nel caso del beacon, se lo sono anche portati a casa.

Elenco delle figure

Elenco delle tabelle

Indice analitico

Bibliografia

[1] Kotlin: https://it.wikipedia.org/wiki/Kotlin_(linguaggio_di_programmazione)

[2] Beacon: https://www.nearit.com/it/cosa-sono-i-beacon/

[3] CentOS vs Ubuntu:
http://www.html.it/articoli/centos-vs-ubuntu-distribuzioni-server-a-confronto/

[4] NGINX: https://it.wikipedia.org/wiki/Nginx

[5] Bluetooth: https://it.wikipedia.org/wiki/Bluetooth

[6] NFC: https://it.wikipedia.org/wiki/Near_Field_Communication

[7] openHAB: https://www.openhab.org/docs/

[8] IoT: https://it.wikipedia.org/wiki/Internet_delle_cose

[9] ARERA: https://www.arera.it/it/operatori/smartmetering.htm

[10] IoTinitalia:https://blog.osservatori.net/it_it/mercato-iot-in-italia

[11] IoT e Big Data:
https://www.thenextfactory.it/2017/09/progettare-sistema-raccolta-analisi-dati/

[12] Tecnologie usate nelle industrie italiane:
https://www.zerounoweb.it/analytics/big-data/tecnologie-per-big-data-analytics/

[13] Python: https://it.wikipedia.org/wiki/Python

[14] Realm vs SQLite:
https://expertise.jetruby.com/choosing-database-for-android-realm-vs-sqlite-51c90e83bafd

[15] Realm:
https://www.cleveroad.com/blog/realm-vs-sqlite-what-is-the-best-database-for-android-app-development

[16] Volley: http://www.html.it/pag/56479/accedere-alla-rete-con-volley/

[17] Gson: https://en.wikipedia.org/wiki/Gson

[18] MongoDB: https://it.wikipedia.org/wiki/MongoDB

[19] Java: https://it.wikipedia.org/wiki/Java_(linguaggio_di_programmazione)

[20] Lista pattern: http://www-sop.inria.fr/marelle/Laurent.Thery/lsp/pattern.html

[21] Template pattern: https://it.wikipedia.org/wiki/Template_method

[22] Estimote: https://estimote.com/

Un ringraziamento

ai miei genitori che mi hanno sempre motivato,

alla mia ragazza che mi è sempre stata vicina

e agli amici con i quali ho condiviso questa splendida esperienza universitaria.