Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica per il Management

Studio dell'applicazione di MySpot: un'App per lo scambio di parcheggi in scenari urbani

Relatore:
Dott.
FEDERICO MONTORI

Presentata da:
QAZIM MUCODEMA

Sessione II
Anno Accademico 2019/2020

Alla mia famiglia,
che mi è sempre stata affianco

Sommario

In [22] è stato stimato che il 70% della popolazione europea vive in centri urbani, i quali sono esposti all'inquinamento dell'aria e del suono. In [7] viene affermato che il tempo speso alla ricerca di un posto di parcheggio può produrre considerevole quantità di inquinamento. Un sistema informativo in grado di assegnare un posto vacante ad una persona in cerca di un parcheggio aiuta la diminuzione dell'inquinamento ed è molto efficace per evitare la congestione del traffico, soprattutto nelle città più grandi. In questo elaborato viene introdotto un applicativo, basato sulla sharing economy, per la risoluzione di questa problematica, dove gli utenti scambiano posti di parcheggio tra di loro, senza l'ausilio di sensori per il riconoscimento di posti disponibili. Attraverso questa tesi verrà illustrata l'architettura della piattaforma, i punti di forza e i punti deboli riguardanti le varie funzionalità implementate.

Indice

Elenco delle figure

Capitolo 1

Stato dell'arte

1.1 Internet of Things

L'Internet of Things, noto anche come Internet degli oggetti, è un paradigma che si basa sulla connettività di dispositivi elettronici. L'Internet Of Things comprende diverse tecnologie che includono: sensori hardware/firmware,e varie tecnologie per la modellazione, l'immagazzinamento, la processazione o la comunicazione di dati. Questo approccio potrebbe fare in modo che i vari oggetti smart [16] che ci circondano, possano essere connessi tra di loro e comunicare attraverso il minimo intervento umano [17]. Un obiettivo importante da raggiungere nel futuro, attraverso lo sviluppo e il miglioramento di queste tecnologie, è quello di riuscire a creare un mondo nel quale gli oggetti smart siano in grado di determinare i nostri piaceri, i nostri bisogni e ad agire di conseguenza senza la necessità di emettere istruzioni esplicite, in modo tale da creare *'un mondo migliore per gli esseri umani'*[12]. Il termine Internet of Things fu inizialmente coniato da Kevin Ashton [6] durante una presentazione nel 1998, in cui venne citato: *"L'Internet of Things ha il potenziale di cambiare il mondo, proprio come Internet ha fatto... forse anche di più"*. Molti ricercatori lavorano in questo campo. Dato che non esiste una definizione standard, introduciamo alcune definizioni proposte da vari ricercatori:

- Definizione da [29]: *"Gli oggetti hanno identità e personalità virtuali che operano in spazi intelligenti, utilizzando interfacce smart per connettersi e comunicare all'interno di contesti sociali e ambientali"*
- *Definizione da [14]: "L'Internet of Things, semanticamente, viene interpretata come una rete internazionale di oggetti interconnessi, indirizzabili in modo univoco, basandosi sui protocolli standard di comunicazione."*

- Definizione da [31]: *"L'Internet of Things permette alle persone e agli oggetti di essere connessi in qualsiasi momento, luogo, con qualsiasi oggetto e con chiunque, idealmente utilizzando qualunque path o network e usufruendo di determinati servizi."*

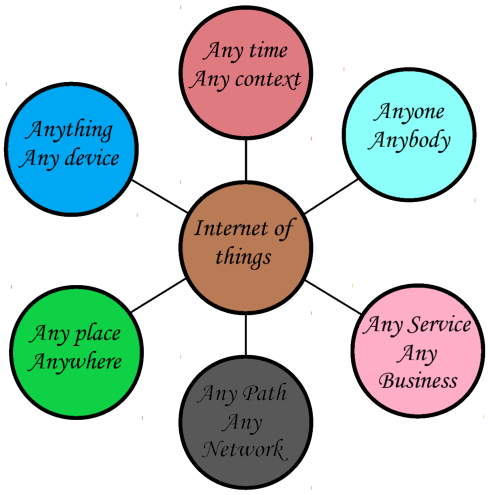

Figura 1.1: L'Internet of Things permette alle persone e agli oggetti di essere connessi in qualsiasi momento, ovunque, con qualsiasi cosa e chiunque, idealmente utilizzando qualsiasi path o network e qualsiasi servizio. [31]

La possibilità di interconnessione e comunicazione tra i vari oggetti, che utilizziamo quotidianamente, permette la possibilità di sviluppare diverse applicazioni, in modo tale da soddisfare le necessità degli utenti. Queste applicazioni si possono suddividere in tre domini principali [4] [28]: applicazioni relative all'industria, applicazioni relative all'ambiente e alla società. Questo insieme di applicazioni, oltre a sfruttare dispositivi che utilizziamo quotidianamente come il pc o lo smartphone, possono utilizzare tutta una serie di strumenti, i quali contengono svariati sensori e sono in grado di compiere determinate funzionalità. Queste reti di dispositivi hanno una particolare importanza in ambito sociale, in quanto facilitano le nostre attività quotidiane; se pensiamo alle "smart home", ambienti dove sono implementati tutti i sensori per fare in modo che, al riconoscimento di determinati gesti o movimenti, venga svolta una determinata azione. Queste reti di dispositivi elettronici possono avere molte applicazioni, tra cui possiamo menzionare l'impatto che hanno avuto dal punto di vista economico le aziende, permettendo un miglioramento dei processi di automazione, distribuzione, sistemistica etc... Al giorno d'oggi,

molte aziende di automobili stanno investendo in queste tecnologie per offrire ai consumatori la migliore esperienza possibile durante l'utilizzo dei loro prodotti. Al giorno d'oggi esistono automobili che attraverso tecnologie di intelligenza artificiale, combinata all'Internet Of Things, non necessitano di una persona al volante, ma riescono ad interagire con il mondo esterno senza provocare incidenti stradali.
Nella Fig.2, vengono espressi le tecnologie, gli standard e i concetti principali che caratterizzano la visione dell'Internet of Things, si può notare che il concetto di IoT può essere considerato come la convergenza tra le tre visioni principali espresse nell'immagine.

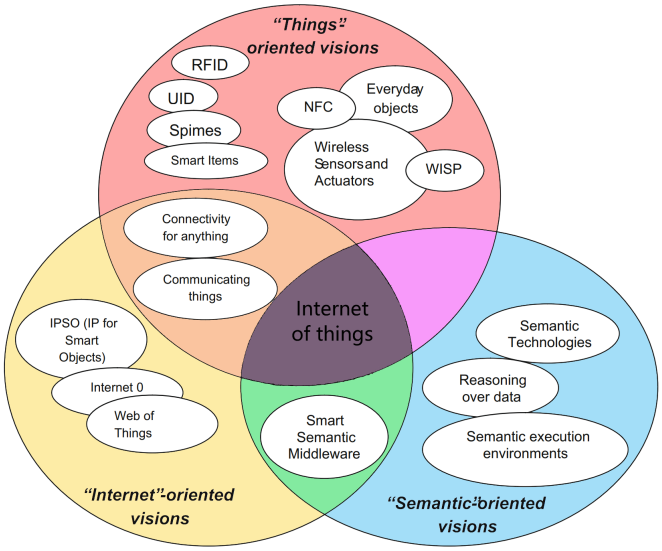

Figura 1.2: Il paradigma 'Internet of Things' come risultato della convergenza di visioni differenti rispetto al concetto. [4]

In seguito analizziamo le caratteristiche principali di questo paradigma, identificate in [31] [23]: *intelligenza, architettura, sistemi complessi, considerazioni sulla dimensione , considerazioni sul tempo, considerazioni sullo spazio, tutto come un servizio.* Durante la progettazione di un sistema IoT, un buon approccio è considerare questi aspetti che in seguito verranno analizzati nello specifico:

- **Intelligenza:** I sensori, che costituiscono una parte importante delle architetture IoT, sono una risorsa importante nella produzione di dati, i quali vengono processati in seguito e hanno una particolare importanza e rilevanza per effettuare l'interconnessione tra i vari oggetti del sistema.

- **Architettura:** L'architettura del sistema dipende anche dagli obiettivi stabiliti e dal problema che si vuole risolvere. Le principali architetture comunemente utilizzate sono due: architettura orientata agli eventi e orientata su un asse temporale. La seconda architettura si basa sulla lettura di dati dai sensori, in modo continuo e in base ad intervalli specifici. La prima invece, all'avvenimento di un evento, viene eseguita la lettura dei dati e le altre operazioni desiderate.

- **Sistemi complessi:** Queste architetture IoT possono comprendere l'utilizzo di una vasta gamma di dispositivi e sensori che interagiscono autonomamente. Bisogna assicurare che quando un dispositivo viene aggiunto o eliminato dalla rete non ci siano problemi riguardanti l'esecuzione dell'applicazione. Nel caso in cui ci siano migliaia di sensori e di dispositivi collegati la continua rimozione e aggiunta di nuovi oggetti dal sistema può rappresentare un problema. Un altra questione che bisogna valutare è il fatto che non tutti gli oggetti hanno la stessa memoria o la stessa potenza di calcolo.

- **Considerazioni sulla dimensione:** L'aumento esponenziale di dispositivi connessi ad Internet, col passare degli anni, deve essere gestito dall' Internet of Things, in modo tale da garantire l'interconnessione e l'esecuzione delle varie interazioni nella rete. Durante la progettazione bisogna considerare questo problema e cercare di costruire l'architettura in modo tale da reggere il continuo aumento dei dispositivi connessi.

- **Considerazioni del tempo:** L'Internet of Things teoricamente deve essere in grado di gestire milioni di eventi paralleli simultaneamente, dato il numero di iterazioni in continuo aumento. In questi sistemi l'elaborazione dei dati in tempo reale diventa essenziale e rappresenta uno strumento potente da utilizzare per fare fronte al carico massivo del sistema.

- **Considerazioni sullo spazio:** Con l'aumento degli oggetti collegati alla rete, il monitoraggio della geolocalizzazione, la quale è strettamente collegata alla computazione context-aware, assume un ruolo importante all'interno di queste reti, dato che le interazioni possono essere vincolate su questo punto di vista, offrendo diverse interazioni in base alla presenza di altri enti, in determinati luoghi, oppure in base all'ambiente che circonda un ente.

- **Everything-as-a-service:** L'accesso alle risorse attraverso un servizio, per mezzo del Platform-as-a-Service (PaaS), Infrastructure-as-aService (IaaS), Software-as-a-Service (SaaS), è altamente diffuso nei

giorni d'oggi. Il modello Everything-as-a-service è caratterizzato dall'efficienza, scalabilità e facilità di utilizzo. Questo modello può essere utilizzato nell'IoT date le sue proprietà in modo tale da passare ad un approccio sensing-as-a-service [33].

La mia applicazione si basa su questo paradigma, dove un utente, attraverso i sensori GPS del telefono, riesce a comunicare la propria posizione agli altri utenti in modo tale da riuscire ad offrire un posto di parcheggio o cercarne uno nelle vicinanze.

1.2 Context Aware Computing

Ci sono molte definizioni in letteratura in merito al Contex-Aware Computing. In [2] un sistema è definito context-aware se utilizza il contesto per fornire informazioni e/o servizi all'utente, dove la rilevanza dipende dalla mansione dell'utente. Prima di continuare bisogna dare una definizione del concetto di "Context", riportiamo la definizione data da [2] dove il contesto viene definito come: *"qulsiasi informazione che può essere utilizzata per caratterizzare la situazione in cui si trova un'entità. Un'entità può essere qualsiasi persona, posto o oggetto, che viene considerato rilevante durante l'interazione tra l'utente e l'applicazione, includendo utenti e altre applicazioni"*. Notiamo che non tutti i dati fanno parte del contesto. Al riguardo Snachez in [25] distingue in due tipologie le categorie dei dati che vengono utilizzati in molte applicazioni nei giorni contemporanei:

- **Raw (sensor) data:** *Qui includiamo la totalità dei dati raccolti non processati da diverse risorse, come ad esempio i sensori*
- **Context Information:** *Queste informazioni vengono generate dall'elaborazione dei raw data.*

In seguito riportiamo un esempio per capire meglio questa distinzione. I dati rilevati dai sensori GPS di uno smartphone vengono considerati come "raw data" dato che senza un'elaborazione, non è possibile recuperare dati importanti che ci diano il contesto, quando viene effettuata la processazione, e siamo in grado di recuperare una posizione geografica come latitudine,longitudine, questa informazione fa parte del contesto dato che può essere poi utilizzata dalle varie applicazioni per raggiungere certi obiettivi. Notiamo che la maggior parte dei dati acquisiti dai sensori, i quali non vengono processati, sono considerati maggiormente come "raw data".

L'acquisizione del contesto viene effettuato attraverso l'utilizzo di sensori specifici che sono suddivisi in 3 sottocategorie:

- "Physical sensors" dai vari dispositivi elettronici.
- "Virtual sensors" che tratta dati recuperati da risorse multiple.
- "Logical sensor" che unisce sensori fisici e virtuali per produrre altre informazioni.

Ora che siamo in grado di determinare se una certa informazione fa parte del contesto, possiamo notare che non tutte le informazioni sono equivalenti o che appartengono alla stessa categoria; abbiamo bisogno di uno schema di caratterizzazione del contesto per classificare i dati recuperati.

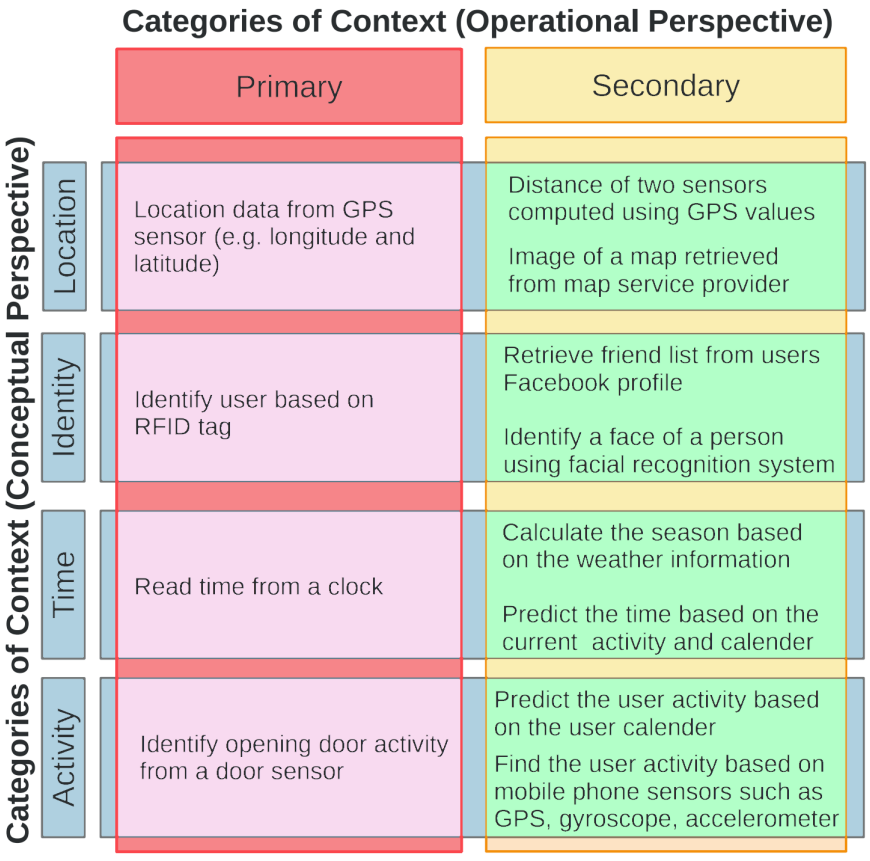

Figura 1.3: Categorizzazione del contesto da due punti di vista differenti: concettuale e operazionale [23].

La figura rappresenta le seguenti definizioni di tipi di contesto [23]: Sorge la necessità di differenziare le tipologie di dati su cui operiamo. Come definito in [23] possiamo distinguere i dati in:

- **Contesto primario:** *Qualsiasi informazione recuperata senza l'ausilio di un contesto esistente o l'esecuzione di operazioni di data fusion.*

- **Contesto secondario:** *Qualsiasi informazione che può essere ritrovata attraverso l'utilizzo del contesto primario da un'operazione di "data fusion". In questa sezione includiamo anche le varie operazioni di "data retrieval" attraverso i servizi web.*

Il contesto è un elemento molto importante per questi sistemi. Ora che sappiamo il suo significato e le categorie dei dati che lo includono, possiamo passare al concetto di sistema "Context-Aware". In seguito riportiamo la definizione di [2]: *"Un sistema è definito context-aware se utilizza il contesto in modo tale da offrire informazioni rilevanti e/o servizi all'utente, quando la rilevanza dipende dalle sue necessità"*. Notiamo anche che, se siamo in grado di identificare quali informazioni fanno parte del contesto e quali no, non è assolutamente scontato il loro utilizzo nelle applicazioni dai programmatori, data la vasta gamma di applicazioni che è possibile implementare. In [26] viene proposta una categorizzazione delle varie applicazioni "context-aware" come segue:

- **Proximate selection:** *Si tratta di una tecnica basata sull'implementazione di un'interfaccia utente, dove gli oggetti localizzati nelle vicinanze vengono enfatizzati o resi facilmente selezionabili.*

- **Automatic contextual reconfiguration:** *Queste applicazioni prevedono processi, nei quali, al cambiamento del contesto, vengono effettuate diverse operazioni come l'aggiunta di un nuovo componente, l'alterazione delle connessioni tra quelli esistenti o la rimozione di un componente dal sistema.*

- **Contextual information and commands:** *Si tratta di applicazioni in grado di produrre risultati diversi in base al contesto in cui si trovano.*

- **Context-triggered actions:** *Si tratta di applicazioni che prevedono clausole "IF-THEN" per specificare come il sistema context-aware debba adattarsi.*

Le categorie sopra menzionate rappresentano le classi delle applicazioni context-aware. Negli anni, oltre a definire le varie classi, si è cercato di determinare una tassonomia sulle proprietà di questi sistemi. In [2] vengono stabilite quali proprietà le applicazioni context-aware possono possedere nelle seguenti categorie: *presentazione* delle informazioni o dei servizi all'utente, *esecuzione* di un servizio, *tagging* del contesto alle informazioni da utilizzare in successivi

richiami. Ritornando alla questione di prima, riguardante le situazioni in cui si trovano i programmatori di queste applicazioni, in [8] viene affermato che, per quanto riguarda le applicazioni mobili, esistono due modi per sfruttare il contesto, riportiamo le definizioni:

- **Context Awareness attiva:** *Quando il contesto viene alterato, l'applicazione automaticamente si adatta cambiando i propri comportamenti.*

- **Context Awareness passiva:** *L'applicazione presenta il contesto nuovo o aggiornato ad un utente interessato, inoltre rende il contesto persistente per future operazioni dell'utente.*

Nell'implementazione di un sistema context-aware è buona norma seguire determinati principi in modo tale da garantire la funzionalità dei componenti a larga scala. In seguito riportiamo gli aspetti più importanti considerati in [23] [20] [24] [5]:

- **Architecture layers and components:** *Le funzionalità del sistema devono essere suddivise in strati organizzati in modo sensato. Una buona norma è fare in modo che ogni componente performi un carico limitato di lavoro, facendo in modo che possa funzionare in modo indipendente.*

- **Scalability and extensibility:** *Il sistema deve prevedere la possibilità da parte di un componente di aggiungersi o lasciare il sistema; questo deve poter essere effettuato in modo efficiente e facile, facendo fronte anche ad un sovraccarico di nuove entrate.*

- **Application programming interface (API):** *Tutte le funzionalità devono essere disponibili e accessibili attraverso un API.*

- **Debugging mechanisms and tools:** *Testare l'affidabilità di un sistema è indispensabile, dato che la maggior parte delle tecnologie sono IoT, questa fase presenta diverse difficoltà dovute alla crescita esponenziale delle interazioni. In merito a questa problematica è buona norma utilizzare meccanismi integrati di debug all'interno del framework.*

- **Automatic context life cycle management:** *I vari framework context-aware devono essere in grado di rilevare in automatico il contesto attraverso il minimo intervento umano possibile.*

- **Context model in-dependency:** *Il contesto deve essere modellato e salvato in uno spazio diverso da quello utilizzato dal framework context-aware per il suo codice e le sue strutture dati, facendo in modo che siano indipendenti tra di loro.*

- **Extended, rich, and comprehensive modelling:** *In un framework context-aware ideale basato sull'IoT, diversi modelli di rappresentazione del contesto devono essere incorporati univocamente, così da migliorare l'efficienza e l'efficacia del sistema.*

- **Multi-model reasoning:** *Diversificare il "model reasoning" aumenta l'efficacia del sistema e abbatte gli svantaggi che ognuno di essi possiede.*

- **Mobility support:** *I framework devono essere sviluppati in modo tale da supportare la totalità dei sensori utilizzati nei vari dispositivi mobili.*

1.3 Sistemi per il Parcheggio smart

Diverse soluzioni sono state proposte per risolvere le problematiche che riguardano il parcheggio nelle città metropolitane, dove il flusso di richieste è molto elevato. La risoluzione di questo problema consegue diversi effetti alla nostra vita quotidiana. Parqex [1] offre un'applicazione basata sulla sharing economy, in cui viene proposto di sfruttare i posti di parcheggio inutilizzati nelle varie proprietà, in modo tale da offrirli alle persone interessate ad un definito orario. In questo modo è possibile usufruire di questi posti vacanti e aumentare la quantità di postazioni disponibili in una determinata area. Altri enti si sono focalizzati nel soddisfare una gestione efficiente dei slot disponibili all'interno di garage privati. In [32] e [13] sono state proposte delle soluzioni in merito che coinvolgono nel primo articolo l'utilizzo di sensori wireless collegati ad un server centrale, facendo in modo che i posti di parcheggio disponibili vengano visualizzati da tutti gli utenti attraverso il web. La seconda soluzione invece prevede la ricognizione attraverso le telecamere della targa della macchina, utilizzando tecniche di "machine learning", e l'associazione ad un utente che fa richiesta di un posto di parcheggio. La qualità della videocamera rispetto a condizioni ambientali come scarsa luminosità nel garage, a volte comporta un errore nella ricognizione della targa, questo può comportare diverse problematiche al sistema dato che non si è in grado di determinare se l'utente sta usufruendo del parcheggio assegnato. Altre soluzioni sono state proposte in associazione con i vari comuni dove, una volta

trovato il posto di parcheggio comunale disponibile, viene effettuato il pagamento della tariffa attraverso un applicativo collegato al sistema comunale, in modo tale che questa informazione sia disponibile ai vari controllori. Si nota che ci sono diverse tipologie di sitemi informativi che tentato di risolvere il problema dei parcheggi. In [11] è stato elabrorato un sondaggio sulle varie tipologie di sistemi di parcheggio smart (SPS). In questo articolo vengono definite 6 categorie principali di SPS: *sistemi per la prenotazione di parcheggio, sistemi informativi relativi alla guida al parcheggio, crowdsourcing in sistemi di trasporto intelligenti, ricerca di parcheggio assistita e centralizzata, sistemi di guida basati su agenti e sistemi di parcheggio per veicoli elettrici.* In seguito analizzeremo le caratteristiche principali di ogni categoria esaminate in [11]:

- **sistemi per la prenotazione di parcheggio:** Questa categoria comprende i sistemi nei quali è possibile prenotare un posto di parcheggio in anticipo e fare in modo che sia disponibile fino al momento dell'arrivo, indicando agli altri utenti quali posti sono stati scelti e quindi non più disponibili. Il sistema si basa sulla comunicazione tra gli utenti in cerca di parcheggio o che hanno precedentemente prenotato e i server che possiedono le informazioni riguardanti ai posti disponibili e occupati. La comunicazione può avvenire in diversi modi, quelli più utilizzati sono le applicazioni sui dispositivi mobili e le pagine web [10] [21]. Questa categoria contiene diverse tipologie di sistemi specializzati in certi aspetti, come sistemi di prenotazione in tempo reale. in questo caso è necessario anche un sistema di monitoraggio in real-time dei posti di parcheggio che si liberano e venogno occupati in continuazione o sistemi di prenotazione, dove ad un utente gli viene assegnato un posto di parcheggio in automatico, in base alla destinazione che desidera raggiungere e la disponibilità di posti di parcheggio in quel momento.

- **Sistemi informativi relativi alla guida al parcheggio:** In questa categoria viene incluso l'insieme dei sistemi informativi che guidano l'utente al raggiungimento di un posto di parcheggio disponibile. Per lo svolgimento di questa operazione, il sistema deve essere in grado di riconoscere lo stato dei posti di parcheggio, per poi notificare all'utente quale tra questi è libero nelle vicinanze. Questa categoria si suddivide ulteriormente in 4 strati:

  - **1.** Il primo strato comprende meccanismi per il monitoraggio del parcheggio che sono in grado di rillevare quando un veicolo arriva o lascia il posto di parcheggio.

  - **2.** Il secondo strato comprende dei meccanismi per la diffusione delle informazioni relativi ai posti di parcheggio disponibili per tutti gli utenti della piattaforma.
  - **3.** Il terzo strato consiste nelle infrastrutture di rete utilizzate per la comunicazione tra tutti gli attori del sistema. In merito a ciò possono essere utilizzate: reti 3G/4G [19], Wi-Fi [18], vehicular ad hoc network (VANET) [30], wireless sensor networks (WSN) [32], o reti wireless ZigBee [27].
  - **4.** Infine, il quarto strato consiste nel centro di controllo che processa l'occupazione dei posti di parcheggio nei vari centri e fa in modo che vengano resi disponibili e visualizzabili per gli utenti della piattaforma in cerca di un posto libero.

- **Crowdsourcing in sistemi di trasporto intelligenti:** Questi sistemi si basano sul concetto di crowd, una communità di motoristi che attraverso i dispositivi mobili, collaborano e aggiornando il sistema con dati in tempo reale, evitando l'utilizzo di sensori sofisticati che rappresentano un costo notevole nelle altre tipologie di sistemi. Si hanno un numero considerevole di utenti che aggiornano il database segnalando posti di parcheggio disponibili, i quali verrano poi trasmessi agli altri utenti che sono in cerca di posti liberi. Inoltre questi sistemi sono in grado di ottenere informazioni in modo passivo, utilizzando la geolocalizzazione dei dispositivi mobili in modo tale da poter determinare quando un utente parcheggia il suo veicolo in tempo reale. Il successo di questi sistemi dipende ampiamente dalla densità degli utenti che contribuiscono al sistema attraverso l'utilizzo delle applicazioni mobili.

- **Ricerca di parcheggio assistita e centralizzata:** Queste tipologie di sistemi prevedono un server centrallizzato che effettua interamente l'elaborazione delle informazioni raccolte dai sensori localizzati nei posti di parcheggio, facendo in modo che vengano resi disponibili ai vari utenti della piattaforma. Di solito questi sistemi non prevedono la possibilità di tracciare la rotta per raggiungere il posto di parcheggio desiderato. L'architettura centralizzata limita la scalabilità del sistema nel caso in cui l'infrastruttura aumenti o cambi nel corso del tempo. Si noti che l'implementazione di tale struttura rappresenta un investimento considerevole oltre poi a considerare i costi relativi alla manutenzione dei vari sensori e del server.

- **Sistemi di guida basati su agenti:** Questi sistemi si basano sul concetto di agente. Un agente può essere qualsiasi modulo di tipo soft-

ware in grado di percepire dati o azioni attraverso sensori ed eseguire un'azione inviando o scambiando informazioni [9]. Questi agenti sono caratterizzati da: autonomia, reattività e adattabilità. Attraverso queste caratteristiche si è in grado di implementare meccanismi automatici che rendono il sistema dinamico e interattivo. In questi sistemi gli agenti collaborano tra di loro scambiando informazioni rillevanti, questo evita un'architettura centralizzata e ottimizza la scalabilità del sistema. La comunicazione degli agenti in termini di determinare quali informazioni sono rillevanti e quali no, porta ad algoritmi complessi, i quali consistono in un costo elevato per questi sistemi, non essendo sempre facili da individuare.

- **Sistemi di parcheggio per veicoli elettrici:** Si tratta di sistemi che aiutano gli utenti in possesso di veivoli elettrici a gestire e pianificare il parcheggio dell'automobile. Durante la fase di pianificazione, vengono valutati aspetti importanti relativi a questa tipologia di automobili; tra queste possiamo menzionare il livello di carica del veicolo, le preferenze dell'utente, l'identificazione dei centri di parcheggio dove è possibile ricaricare il veicolo o il prezzo dell'elettricità dei vari centri [3]. Generalmente questi sistemi offrono i servizi agli utenti attraverso applicazioni mobili o web che possono richiedere informazioni relative allo stato di ricarica del veicolo o il tempo necessario per il parcheggio, facendo in modo che la ricarica venga eseguita in modo efficente, evitando di sovraccaricare la rete elettrica e limitare il numero di posti auto disponibili, massimizzando così lo stato di carica dei veicoli e soddisfacendo le esigenze dei conducenti, ma ache ottimizzando il consumo e la disponibilità della rete elettrica[11].

1.4 Motivazioni

Dopo aver fatto un'attenta revisione della letteratura e dato che non c'è niente di simile in merito; ho deciso di costruire un sistema informativo che riguardi la gestione dei parcheggi in centri abitati, risolvendo così un problema reale nel quale ognuno di noi si ritrova ogni giorno. Questa soluzione può avere un effetto positivo riguardante il tempo necessario che ogni utente impiega alla ricerca di un posto di parcheggio, riducendo in questo modo l'inquinamento e la congestione stradale, prodotta dalla molteplicità di persone in circolazione alla ricerca di un posto libero. Uno dei punti fondamentali della mia applicazione è che non necessita infrastrutture complesse o l'installazione di sensori per individuare i posti di parcheggio disponibili. Essa usufruisce dei disposi-

tivi mobili che utilizziamo ogni giorno e di un database attraverso il quale è possibile effettuare richieste e offerte di parcheggio. Da un certo punto di vista i costi relativi alle varie strutture sensoristiche vengono abbattute e sono pari a 0. All'interno della mia applicazione utilizzo informazioni che fanno parte del contesto, nel mio caso informazioni relative alla geolocalizzazione, in una situazione più evoluta mi immagino un caso in cui l'app, attraverso l'activity recognition, è in grado di riconoscere se un utente stia guidando. Il progetto sviluppato si basa sul concetto di sharing economy, un modello economico alternativo al consumismo classico, basato su pratiche di scambio e di condivisione alla pari tramite una piattaforma informatica. Un utente che è in cerca di parcheggio effettua una richiesta attraverso l'applicativo e in seguito li verrà assegnato uno slot libero nelle sue vicinanze. Attraverso la revisione delle varie tipologie di soluzioni a questo problema, è possibile inquadrare la soluzione presentata nella categoria *Crowdsourcing in sistemi di trasporto intelligenti*, essendo che si basa sul concetto di crowd, di utenti che segnalano i posti di parcheggio disponibili i quali vengono assegnati successivamente ad utenti in ricerca. Ritengo che la mia applicazione sia innovativa in quanto, dalla revisione effettuata, viene introdotta per la prima volta un sistema basato su biglietti per effettuare lo scambio di un posto di parcheggio.

Capitolo 2

Architettura

2.1 Descrizione generale

Abbiamo analizzato durante lo scorso capitolo la necessità di un sistema di monitoraggio efficente dei posti di parcheggio. Considerando la potenza dei dispositivi mobili al giorno d'oggi, è stato deciso di utilizzarli come strumento principale di questa applicazione. Con l'avanzamento della tecnologia si presume che le automobili diventeranno sempre più intelligenti e in grado di effettuare operazioni complesse, per questo motivo nel futuro il campo di operazione di questa applicazione potrà scalare dal telefono all'automobile. L'applicazione è stata nominata *MySpot* in modo tale da essere intuitivo per l'utente il fatto che essa gestisca il parcheggio del veicolo. Oltre alle funzionalità principali di offerta e richiesta di un posto di parcheggio, è stato pensato di offrire un quadro completo per la gestione del parcheggio di una macchina. Ai fini di ampliare le funzionalità offerte all'utente, sono state aggiunte due funzionalità: quella di salvare un posto di parcheggio e quella di tracciare un itinerario per raggiungere l'automobile parcheggiata. In questo modo oltre a risolvere la problematica dei parcheggi, l'utente non rischia di dimenticare dove è situata la sua automobile, una volta salvata la posizione, attraverso l'applicativo, è facile individuare la sua posizione e raggiungerla, riducendo in questo modo il tempo necessario per arrivare al posto di parcheggio dove il veicolo è situato attraverso una navigazione che calcola il tragitto più veloce al punto di arrivo.In seguito presentiamo il diagramma delle attività dell'applicazione:

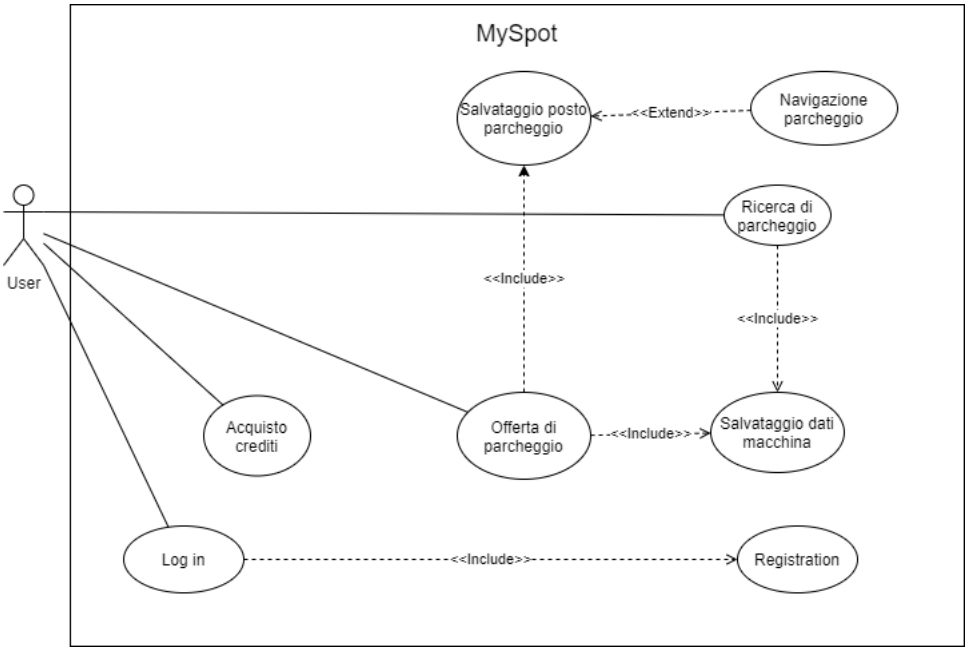

Figura 2.1: Diagramma delle attività dell'applicazione MySpot.

Analizziamo i vari elementi del diagramma per comprendere meglio le funzionalità dell'applicazione:

- **Log in:** *Per gli utenti già registrati è possibile accedere alle funzionalità dell'applicazione effettuando l'accesso attraverso le proprie credenziali: email e password.*

- **Registrazione:** *Per gli utenti che si sono precedentemente registrati o che desiderano creare un nuovo account, è possibile effettuare la registrazione completando le informazioni del proprio profilo.*

- **Acquista tickets:** *L'utente può effettuare l'acquisto di tickets che verranno utilizzati successivamente per effettuare lo scambio dei posti di parcheggio.*

- **Salvataggio posto parcheggio:** *Attraverso questa funzione l'utente, tramite i sensori dello smartphone, è in grado di rilevare la propria posizione attuale, e salvarlo all'interno dell'applicazione come posto di parcheggio. Quindi l'utente salva la posizione dell'automobile.*

- **Navigazione parcheggio:** *Attraverso questa funzionalità è possibile per l'utente seguire un itinerario dalla posizione attuale all'ultimo posto di parcheggio salvato.*

- **Offerta di parcheggio:** *L'utente all'interno dell'applicazione può offrire un posto di parcheggio, effettuando questa operazione nel momento in cui il parcheggio viene scambiato, l'utente avrà una ricompensa per il tempo investito nell'offrire una postazione nella forma di un ticket che potrà successivamente utilizzare per cercare parcheggio.*

- **Ricerca di parcheggio:** *L'utente, al costo di due tickets, può cercare un posto di parcheggio nelle vicinanze e il sistema, nel caso ci siamo disponibilità, gli assegna uno ed effettua la navigazione per il posto di parcheggio.*

- **Salvataggio dati macchina:** *Per effettuare lo scambio di un posto di parcheggio è necessario conoscere l'automobile dell'utente con cui si vuole effettuare lo scambio. In merito a questo aspetto l'applicazione prevede la possibilità di inserire e salvare i dati delle proprie automobili e nel momento in cui si vuole effettuare un'operazione di ricerca o di offerta, selezionare l'automobile in uso.*

2.2 Profilazione

Per risolvere problemi relativi al parcheggio è necessario principalmente distinguere i vari utenti che utilizzano la piattaforma e le varie automobili che hanno a disposizione. A questo scopo viene effettuata una registrazione quando l'utente accede per la prima volta con il suo dispositivo. Attraverso la registrazione vengono salvati i dati principali dell'utente come nome, data di nascita numero di telefono ecc. Un altro aspetto importante riguarda la profilazione delle varie automobili che l'utente utilizza e ha a disposizione. Questo è stato pensato per evitare che l'utente inserisca i dati relativi all'automobile ogni volta che effettua un'operazione di ricerca o offerta di un posto di parcheggio. In questo modo è possibile selezionare l'automobile in uso tra tutte le automobili precedentemente salvate. Si è deciso che per la profilazione di un'automobile vengano salvati i dati relativi al: colore, targa, modello e marchio. Tutti questi dati aiutano a rendere l'automobile visivamente riconoscibile da un altro utente in modo tale da facilitare lo scambio. Inoltre l'utente ha la possibilità di aggiungere una foto per ogni singola automobile. Si è pensato che molte persone al volante, nel momento in cui cercano parcheggio e guidano, non abbiano la possibilità di leggere attentamente tutti i dati relativi al veicolo da identificare, in merito a questa problematica la foto potrebbe aiutare a facilitare il processo in modo tale da rendere l'operazione il più semplice possibile. Un altro punto di vista rilevante riguarda la mancata abilità della maggior parte delle persone, escludendo gli appassionati,

ad identificare le tipologie di auto nel mercato, una foto aiuterebbe ad abbattere queste barriere. L'aggiunta della foto nella piattaforma è facoltativa, però consigliata, dati i benefici che implica. Un altro aspetto interessante è la sezione dell'acquisto di biglietti per lo scambio di posti di parcheggio. Inizialmente è stato pensato di lasciare inserire all'utente il numero di biglietti di cui ha bisogno, per poi proseguire all'acquisto. Poi si è deciso di continuare per una strada diversa seguendo il modello che molti videogiochi hanno messo in atto, cioè creare dei pacchetti standard, con un determinato numero di biglietti acquistabili in base alle proprie necessità. I pacchetti al momento sono 4: pacchetti da 10, 20, 50 e 100 biglietti. Si nota che senza biglietti non è possibile partecipare allo scambio di posti di parcheggio e che l'offerta di un posto libero ha come vantaggio il guadagno di un biglietto che è possibile in seguito utilizzare, se in possesso di un altro biglietto, alla ricerca di un posto di parcheggio.

2.3 Ricerca per un posto di parcheggio

L'utente, per avviare una richiesta di parcheggio, seleziona inizalmente il veicolo che sta utilizzando, il quale viene salvato nel database e verrà poi inviato all'offerente per l'identificazione dell'automobile con la quale avverrà lo scambio. Una volta che questa operazione viene completata con successo, l'utente diventa un richiedente attivo del sistema, ora si attende che ci sia un offerente attivo nelle vicinanze per assegnargli il posto di parcheggio. Una volta individuato, il richiedente visualizza i dati riguardanti l'utente, il veicolo che sta utilizzando e la posizione del parcheggio disponibile. In questo momento, in base a determinati fattori come la distanza, o la destinazione che l'utente intende raggiungere, può scegliere se accettare o rifiutare lo scambio. Nel caso in cui esso decida di rifiutare il posto di parcheggio, l'associazione tra richiedente e offerente viene eliminata e si avvia nuovamente la ricerca. Se l'utente accetta, lo scambio viene comfermato e l'utente inizia la navigazione verso il punto in cui si trova il parcheggio. Una volta raggiunto, si possono avere due situazioni: la prima situazione rappresenta il caso in cui l'offerente non abbia atteso l'arrivo dell'utente, il parcheggio si sia liberato e immediatamente occupato da un'altra persona, in questo caso l'utente, attraverso l'applicazione, avvisa il sistema confermando lo scambio con un esito negativo. In seguito i crediti degli utenti verranno aggiornati rispettivamente. La seconda situazione è quella in cui l'offerente abbia atteso e lo scambio viene effettuato con successo. In questo caso l'utente afferma l'avvenimento dello scambio e in seguito i biglietti di entrambi gli utenti verranno aggiornati con successo. Riportiamo il diagramma di sequenza della richiesta:

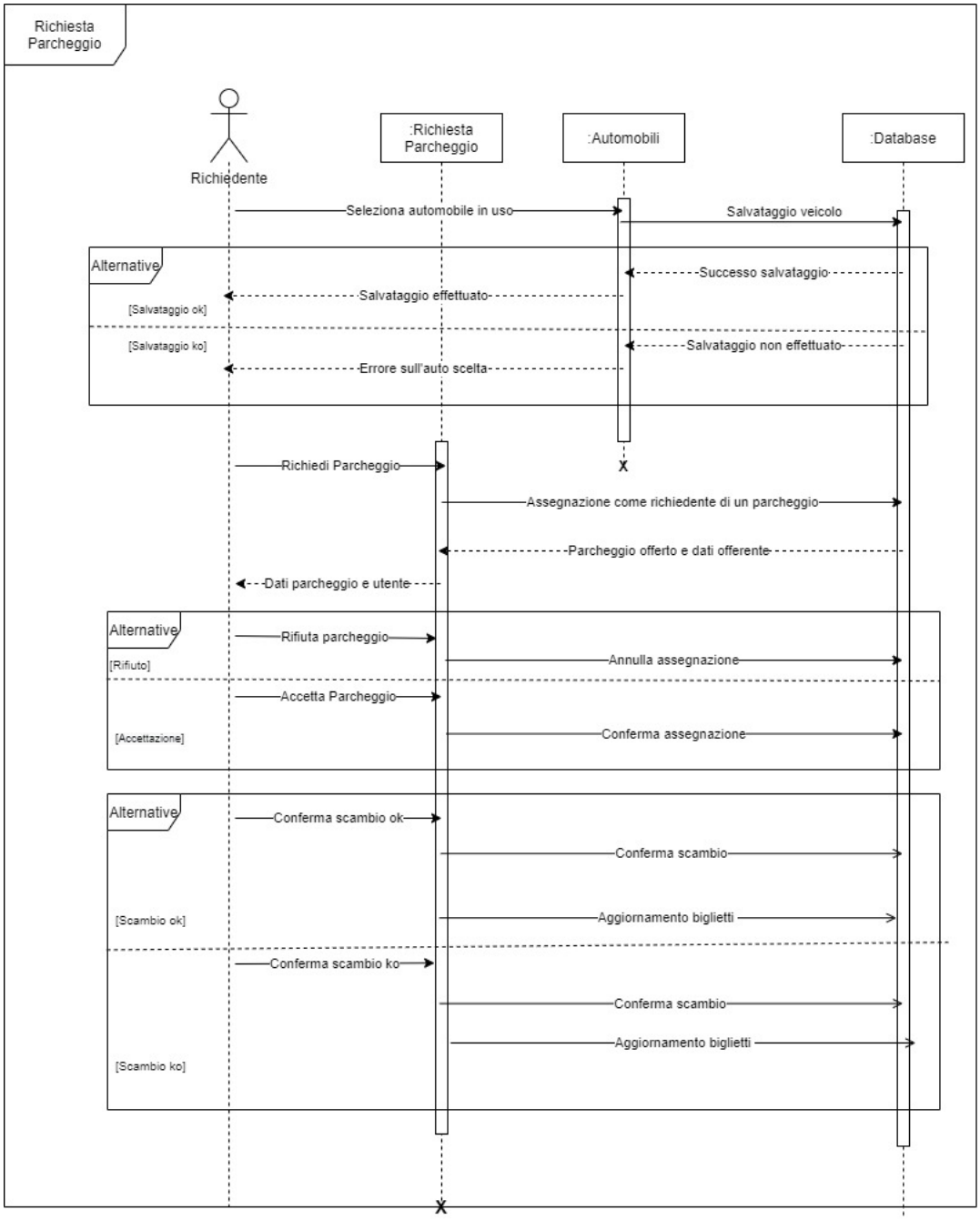

Figura 2.2: Diagramma di sequenza della richiesta di parcheggio.

2.4 Offerta di un posto di parcheggio

L'utente, per avviare l'offerta di un posto di parcheggio, deve inizialmente selezionare tra la lista dei veicoli in possesso quello che sta utilizzando. I dati relativi all'automobile verranno successivamente salvati nel database in modo tale che il richiedente possa accederci. Una volta svolta questa operazione con successo, l'utente richiede un utente in cerca di parcheggio nelle vicinanze ed effettua l'assegnazione attraverso una proposta di parcheggio che può in seguito venire approvata o rifiutata. Nel caso in cui venga rifiutata l'utente cercherà di effettuare nuovamente una proposta ad un altro utente del sistema. Nel caso in cui invece l'utente accetti la proposta, l'associazione viene confermata e l'utente si mette in stato di attesa del richiedente. L'offerente, in seguito all'arrivo dell'utente, viene notificato se esso abbia confermato l'avvenimento dello scambio o meno. In seguito viene riportato il diagramma di sequenza dell'offerta di parcheggio:

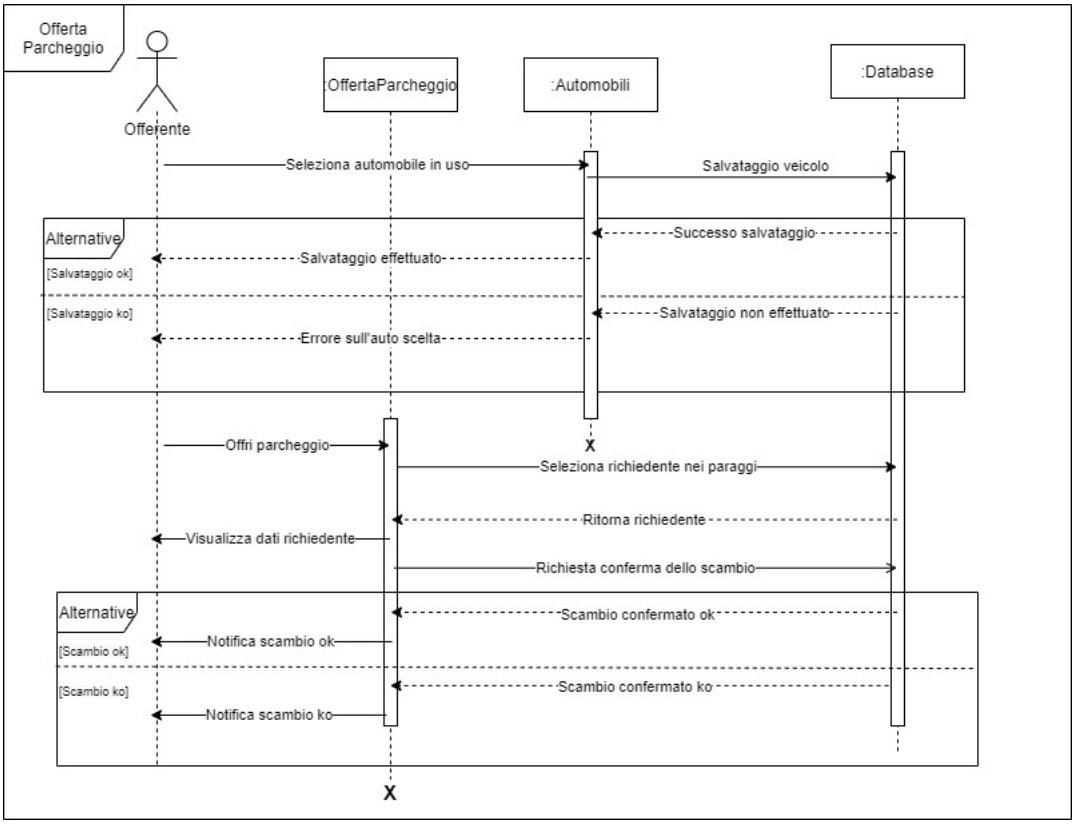

Figura 2.3: Diagramma di sequenza dell'offerta di parcheggio.

2.5 Sitema dei tickets

Lo scambio di un posto di parcheggio tra due utenti pone un problema per quanto riguarda la determinazione dei benefici di entrambe le parti nel processo. Nel caso del richiedente, il beneficio è individuato come il posto di parcheggio ottenuto. Dall'altra parte però bisogna capire cosa potrebbe incentivare un utente ad attendere e offrire il proprio posto di parcheggio. Inizialmente si era pensato a una retribuzione diretta monetaria, quindi l'utente in cerca paga l'utente che offre in modo tale che entrambi siano soddisfatti dallo scambio. Un accorgimento molto importante di questo modello è stato individuato e rappresenta la possibilità da parte degli utenti di guadagnare offrendo parcheggi di continuo, rappresentando una fonte di guadagno per gli offerenti. Questa soluzione potrebbe avere degli impatti positivi sul sistema aumentando l'offerta dei posti di parcheggio, ma allo stesso tempo si voleva evitare di dare la possibilità alle persone di arricchirsi occupando di continuo posti di parcheggio e venderli sulla piattaforma. Per questo motivo è nata la necessità di utilizzare un altro sistema per lo scambio dei parcheggi. In tale proposito è stato ideato uno schema di scambio in base a biglietti / crediti. Per evitare le problematiche sopra menzionate, l'utente che offre parcheggio, invece di acquisire una retribuzione diretta, riceve un quantitativo di biglietti che potrà successivamente utilizzare per cercare un posto di parcheggio. A tale proposito, si noti che il problema di determinare quanto costa un posto di parcheggio, si è traslato nel determinare quanti biglietti sono necessari per lo scambio e quanto costa un singolo biglietto. Al riguardo non è stato effettuato uno studio del mercato per determinare il modello dello scambio dei biglietti, ma si è ritenuta valida la seguente soluzione: Ad ogni transazione di parcheggio, l'offerente deve essere in possesso di 1 credito, mentre il richiedente di 2 crediti. Con il termine credito mi riferisco al ticket virtuale dell'applicativo. Valutiamo i vari casi per cui questo sistema funziona:

- **1.** Nel caso in cui la transazione viene effettuata con successo, il richiedente di parcheggio perde definitivamente i 2 crediti. L'offerente dall'altra parte ottiene 1 credito aggiuntivo oltre a quello investito inizialmente. Da ogni transazione andata con successo viene perso 1 credito, questo garantisce che al passare del tempo ci sia bisogno di acquistare crediti. Risultato finale: richiedente $2 \rightarrow 0$, offerente $1 \rightarrow 2$
- **2.** L'offerente decide di non attendere il richiedente e libera il posto di parcheggio. In questo caso l'offerente perde il suo credito, il quale viene trasferito come bonus al richiedente per essersi spostato ed essere intenzionato allo scambio. Risultato finale: richiedente $2 \rightarrow 3$, offerente $1 \rightarrow 0$.

- **3.** Il richiedente decide di non raggiungere la posizione dell'offerente. In questo caso lui perde entrambi i crediti che vengono trasferiti all'offerente come bonus per l'attesa, si noti che successivamente l'offerente può inizializzare un'altra transazione per lo stesso posto di parcheggio. Risultato finale: richiedente $2 \rightarrow 0$ offerente $1 \rightarrow 3$.

Questo sistema si basa sull'equilibrio di Nash, sia l'utente che offre e sia quello che cerca parcheggio hanno un guadagno dall'avvenimento dello scambio, l'utente che offre parcheggio guadagna un ticket e in seguito può continuare con le proprie attività quotidiane, mentre l'utente che cerca parcheggio, riesce a parcheggiare la macchina. Attraverso uno studio statistico del mercato in futuro, sarà possibile quantificare il costo del singolo biglietto.

Capitolo 3

Implementazione

3.1 Scelte implementative

Per lo sviluppo dell'applicazione è stato scelto l'utilizzo di Firebase Realtime Database. Si tratta di un cloud-hosted database dove i dati vengono salvati in formato JSON e vengono sincronizzati in tempo reale ad ogni utente connesso. La seguente figura rappresenta l'organizzazione della base di dati utilizzata nel progetto.

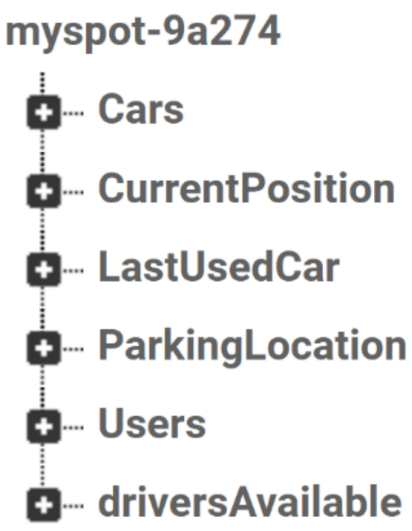

Figura 3.1: Struttura della base di dati in Firebase.

Per il salvataggio delle immagini delle automobili è stato utilizzato Cloud Storage, uno strumento potente per il salvataggio di varie tipologie di file: audio, video o immagini. In seguito viene mostrato l'utilizzo dell'API per il salvataggio delle immagini sulla piattaforma:

```
StorageReference filePath = FirebaseStorage.getInstance().
    getReference().child("car_image").child(user_id);
Bitmap bitmap = null;
try {
    bitmap=MediaStore.Images.Media.getBitmap(getContext().
    getContentResolver(), resultUri);
}catch (IOException e){
    e.printStackTrace();
}
//compressing the image
ByteArrayOutputStream baos = new ByteArrayOutputStream();
bitmap.compress(Bitmap.CompressFormat.JPEG, 20, baos);
byte[] data = baos.toByteArray();
UploadTask uploadTask = filePath.putBytes(data);
uploadTask.addOnSuccessListener(new OnSuccessListener<
    UploadTask.TaskSnapshot>() {
    @Override
    public void onSuccess(UploadTask.TaskSnapshot
    taskSnapshot) {
        if (taskSnapshot.getMetadata() != null) {
            if (taskSnapshot.getMetadata().getReference() !=
    null) {
                Task<Uri> result = taskSnapshot.getStorage().
    getDownloadUrl();
                result.addOnSuccessListener(new
    OnSuccessListener<Uri>() {
                    @Override
                    public void onSuccess(Uri uri) {
                        String imageUrl = uri.toString();
                        Map newImage = new HashMap<>();
                        newImage.put("imageUri",imageUrl);
                        current_user_db.updateChildren(
    newImage);
                    }
                });
            }
        }

    }
});
```

Listing 3.1: Salvataggio delle immagini su Cloud Firebase Storage

Una volta salvate le immagini in Cloud Firestone è nata la necessita di uno strumento per caricare queste immagini nei vari ImageView, a questo scopo

è stato utilizzato Glide, un potente framework open source per la gestione e il caricamento di file multimediali[15].

```
public void setCarImage(final Uri downloadUri){

    Glide.with(mView.getContext()).load(downloadUri).into(
    carImage);

}
```

Listing 3.2: Utilizzo di Glide per il caricamento delle immagini

Ci furono diversi tentativi per il calcolo della distanza tra due utenti della piattaforma, inizialmente si era pensato di utilizzare l'API di Google Maps Direction, studiando l'API, si è notato che per ogni relazione tra due utenti era necessario scaricare un file JSON con molte altre caratteristiche oltre alla distanza e al tempo che ci si impiega per andare da un punto all'altro. Il problema principale è che per una quantità sempre in aumento di utenti, la gestione di tutti questi file diventa dispendiosa. Un secondo problema di carattere economico, riguardava il costo per usufruirne. Durante la ricerca di altre soluzioni, è stato notato Geofire, uno strumento molto utile che può essere utilizzato sui database di Firebase per effettuare query su tuple di tipo GeoLocation, contenenti latitudine e longitudine. Un elemento molto importante che ha fatto sì che l'implementazione continuasse attraverso tale strumento, fu il fatto che le query non vengono eseguite sull'intero insieme di tuple, ma parzialmente, in base alla località specifica, questo ha un impatto positivo sulla scalabilità della piattaforma.

In seguito verranno analizzati i componenti principali del programma:

- **Autenticazione:** Con Autenticazione si intendono le componenti di Login e di registrazione della piattaforma che fanno in modo che un utente si identifichi per continuare ad usufruire delle varie funzionalità dell'applicativo.

- **MenuActivity:** Questa componente comprende il menù dell'applicativo, lo scorrimento tra la lista dei fragments e la richiesta di alcuni permessi se non precedentemente consentiti. Inoltre viene visualizzato il nome e il cognome dell'utente.

- **AccountFragment:** Attraverso questa componente è possibile modificare il proprio profilo.

- **CarFragment:** Questa componente viene utilizzata per la gestione dei veicoli dell'utente, è prevista la possibilità di aggiungere o di eliminare automobili esistenti.

- **TicketsFragment:** Attraverso questa componente, l'utente, se a scorta di biglietti, può effettuare il loro acquisto.

- **HomeFragment:** Attraverso questa componente è possibile usufruire delle funzionalità principali della piattaforma che comprendono la ricerca, offerta, salvataggio e navigazione al posto di parcheggio.

- **DriverMapFragment:** Questa componente prevede tutte le operazioni in merito all'utente come ricchiedente di un posto di parcheggio libero nelle sue vicinanze.

- **OffererMapActivity:**Questa componente invece, riguarda tutte le operazioni relative all'utente come proponente di un posto di parcheggio.

- **TrackingService:** Attraverso questa componente, l'utente che richiede posto di parcheggio è in grado di visualizzare il veicolo dell'utente con cui effettuerà lo scambio e di confermare il suo avvenimento.

3.2 Autenticazione

Quando un utente inizializza l'applicazione, visualizza uno Splash Screen contenente il logo dell'applicazione con un'animazione come mostrato successivamente:

```
new Handler().postDelayed(new Runnable() {
    @Override
    public void run() {
        Intent intent = new Intent(SplashActivity.this,
   LogInActivity.class);
        Pair[] pairs = new Pair[2];
        pairs[0] = new Pair<View,String>(mImage,"logo_image")
   ;
        pairs[1] = new Pair<View,String>(logo,"logo_text");
        ActivityOptions options = ActivityOptions.
   makeSceneTransitionAnimation(SplashActivity.this, pairs);
        startActivity(intent,options.toBundle());
        finish();
    }
},SPLASH_SCREEN);
```

Listing 3.3: Animazione dello Splash Screen

In seguito allo splash screen l'utente viene reindirizzato all'attività di Login dove viene verificato se ha precedentemente effettuato l'accesso. Per fare ciò

è stato utilizzato il servizio di autenticazione offerto da Firebase. In seguito riportiamo il codice che comprende questo controllo.

```
mAuth = FirebaseAuth.getInstance();
fireBaseAuthListener = new FirebaseAuth.AuthStateListener() {
    @Override
    public void onAuthStateChanged(@NonNull FirebaseAuth
    firebaseAuth) {
        FirebaseUser user = FirebaseAuth.getInstance().
    getCurrentUser();
        if(user != null){
            Intent intent = new Intent(LogInActivity.this,
    MenuActivity.class);
            startActivity(intent);
            finish();
            return;
        }
    }
};
```

Listing 3.4: Verifica se le credenziali sono state immesse precedentemente.

Nel caso in cui l'utente abbia precedentemente inserito le credenziali, il sistema lo riconosce e lo reindirizza alla pagina principale dell'applicazione, dove è possibile usufruire di tutti i servizi disponibili. Nel caso contrario invece, ha la possibilità di inserire le credenziali e autenticarsi all'interno dell'applicazione attraverso la seguente funzione:

```
private void signInUser(String email, String password) {
    mAuth.signInWithEmailAndPassword(email, password).
    addOnCompleteListener(LogInActivity.this, new
    OnCompleteListener<AuthResult>() {
        @Override
        public void onComplete(@NonNull Task<AuthResult> task
    ) {
            if (!task.isSuccessful()) {
                Toast.makeText(LogInActivity.this, R.string.
    login_error, Toast.LENGTH_SHORT).show();
            }
        }
    });
}
```

Listing 3.5: Autenticazione al sistema

Nell'eventualità che l'utente non possieda delle credenziali per accedere ai servizi della piattaforma; è possibile effettuare la registrazione inserendo i dati necessari e successivamente usufruire delle funzionalità dell'applicazione. In seguito riportiamo la funzione per la registrazione:

```
private void RegistrationUserWithEmailAndPassword() {
    if(errorMsg.equals("")){
        mAuth.createUserWithEmailAndPassword(email,password).
    addOnCompleteListener(RegistrationActivity.this, new
    OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult>
    task) {
                if(!task.isSuccessful()){
                    Toast.makeText(RegistrationActivity.this,
     "sign up error", Toast.LENGTH_SHORT).show();
                }else{
                    String user_id = mAuth.getCurrentUser().
    getUid();
                    DatabaseReference current_user_db =
    FirebaseDatabase.getInstance().getReference().child("Users
    ").child(user_id);
                    User newUser = new User(email,name,
    surname, dateOfBirth, number);
                    current_user_db.setValue(newUser.
    getUserToMap());
                }
            }
        });
    }else
    {
        Toast.makeText(RegistrationActivity.this, errorMsg,
    Toast.LENGTH_SHORT).show();
    }
}
```

Listing 3.6: Registrazione al sistema.

3.3 MenuActivity

In seguito all'autenticazione, l'utente viene reindirizzato al MenuActivity dove viene inizialmente verificato se l'applicazione possiede i diritti per l'accesso ai dati che riguardano la geolocalizzazione del dispositivo. In caso negativo viene richiesto all'utente di consentire l'accesso. All'interno del menu vengono visualizzati alcuni dati relativi all'utente, per questo motivo è necessario effettuare una lettura dei medesimi dati sul database. In seguito riportiamo i metodi principali dell'activity:

```
    private void getUserDetails() {
        String user_id = mAuth.getCurrentUser().getUid();
        final FirebaseDatabase database = FirebaseDatabase.
    getInstance();
```

```
4       DatabaseReference myRef = database.getReference("
    Users").child(user_id);
5       myRef.addValueEventListener(new ValueEventListener()
    {
6           @Override
7           public void onDataChange(DataSnapshot
    dataSnapshot) {
8               currentUser = dataSnapshot.getValue(User.
    class);
9               mEmail.setText(currentUser.getEmail());
10              mNameSurname.setText(currentUser.getName()+"
    "+currentUser.getSurname());
11              mTickets.setText(String.valueOf(currentUser.
    getTicketCount()));
12          }
13
14          @Override
15          public void onCancelled(DatabaseError
    databaseError) {
16              System.out.println("The read failed: " +
    databaseError.getCode());
17          }
18      });
19   }
20   private void buildAlertMessageNoGps() {
21      final AlertDialog.Builder builder = new AlertDialog.
    Builder(this);
22      builder.setMessage("This application requires GPS to
    work properly, do you want to enable it?")
23              .setCancelable(false)
24              .setPositiveButton("Yes", new DialogInterface
    .OnClickListener() {
25                  public void onClick(@SuppressWarnings("
    unused") final DialogInterface dialog, @SuppressWarnings("
    unused") final int id) {
26                      Intent enableGpsIntent = new Intent(
    android.provider.Settings.ACTION_LOCATION_SOURCE_SETTINGS)
    ;
27                      startActivityForResult(
    enableGpsIntent, PERMISSIONS_REQUEST_ENABLE_GPS);
28                  }
29              });
30      final AlertDialog alert = builder.create();
31      alert.show();
32   }
```

Listing 3.7: Metodi principali del Menù Activity.

3.4 AccountFragment

Attraverso questa componente diamo la possibilità all'utente di alterare i propri dati precedentemente salvati in fase di registrazione. Si tratta di fare in modo che l'utente possa visualizzare i dati precedentemente salvati e di poterli modificare in modo opportuno.

```
    myRef.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            currentUser = dataSnapshot.getValue(User.class);
            mName.setText(currentUser.getName());
            mSurname.setText(currentUser.getSurname());
            mMobilePhone.setText(currentUser.getCel());
            mDateOfBirth.setText(currentUser.getDateOfBirth()
);
        }

        @Override
        public void onCancelled(DatabaseError databaseError)
{
            System.out.println("The read failed: " +
databaseError.getCode());
        }
    });
```

Listing 3.8: Metodo principale dell'Account Fragment

3.5 CarFragment

La necessità di distinguere l'automobile con la quale si intende intercambiare il posto di parcheggio, ha portato alla possibilità di salvare, all'interno dell'applicazione, le tipologie di automobili possedute, specificando le seguenti caratteristiche: targa, modello, marchio e colore. Questo viene effettuato attraverso un RecyclerView, il quale contiene la lista di tutte le automobili aggiunte dall'utente. Ogni elemento del RecyclerView, oltre alle caratteristiche sopra menzionate, può contenere anche una foto, se l'utente l'ha precedentemente salvata, in caso contrario viene mostrata l'immagine standard di un automobile blue. Di conseguenza è stato implementato l'eventuale CarAdapter con la lettura dei dati da Firebase Real Time Database e le foto delle automobili da Firebase Cloud Storage. Nella schermata appare anche un Button che riguarda la possibilità di aggiungere una nuova automobile nella lista. Premendo l'utente verrà reindirizzato al NewCarFragment dove li sarà possibile procedere con l'inserimento dei dati e l'aggiornamento del databa-

se. Per quanto riguarda l'eliminazione delle automobili non più possedute o erroneamente aggiunte, lo scorrimento di un elemento del RecyclerView, verso destra o sinistra, fa in modo che l'automobile venga eliminata.

```
    @Override
    public View onCreateView(LayoutInflater inflater,
ViewGroup container,
        Bundle savedInstanceState) {
        View view=inflater.inflate(R.layout.fragment_car,
container, false);
        mAuth = FirebaseAuth.getInstance();
        mCarList=view.findViewById(R.id.ListOfCars);
        mCarList.setHasFixedSize(false);
        mCarList.setLayoutManager(new LinearLayoutManager(
getContext()));
        setUpRecyclerView();
        new ItemTouchHelper(new ItemTouchHelper.
SimpleCallback(0,ItemTouchHelper.LEFT |ItemTouchHelper.
RIGHT) {
            @Override
            public boolean onMove(@NonNull RecyclerView
recyclerView, @NonNull RecyclerView.ViewHolder viewHolder,
    @NonNull RecyclerView.ViewHolder target) {
                return false;
            }
            @Override
            public void onSwiped(@NonNull RecyclerView.
ViewHolder viewHolder, int direction) {
                adapter.deleteItem(viewHolder.
getAdapterPosition());
            }
        }).attachToRecyclerView(mCarList);

        addCar=view.findViewById(R.id.button_add_car);
        addCar.setOnClickListener(new View.OnClickListener()
{
            @Override
            public void onClick(View v) {
                getActivity().getSupportFragmentManager().
beginTransaction().replace(R.id.fragmentContainer, new
NewCarFragment()).commit();
            }
        });

        return view;
    }
    private void setUpRecyclerView(){
        String user_id = mAuth.getCurrentUser().getUid();
```

```
33          Query query = FirebaseDatabase.getInstance().
    getReference().child("Cars").child(user_id);
34          FirebaseRecyclerOptions<Car> options =
35                  new FirebaseRecyclerOptions.Builder<Car>()
36                          .setQuery(query, Car.class)
37                          .build();
38          adapter = new CarAdapter(options);
39
40          mCarList.setAdapter(adapter);
41      }
```

Listing 3.9: Metodo di CarFragment.

3.6 TicketsFragment

Questa componente viene utilizzata per acquistare i tickets per effettuare lo scambio di posti di parcheggio. Si noti che attualmente non è previsto nessun metodo di pagamento per l'incremento di questi biglietti come Paypal etc. Al momento si tratta unicamente di aggiornamenti sul database della quota di biglietti posseduta da ogni utente. Il seguente fragment contiene all'interno 4 CardViews per cui è possibile effettuare diversi aggiornamenti in base alle proprie necessità. Nel caso in cui si implementasse un metodo di pagamento, il prezzo per i vari aggiornamenti dipenderebbe dalla quantità di biglietti acquistati.

```
1 buy10.setOnClickListener(new View.OnClickListener() {
2             @Override
3             public void onClick(View v) {
4                 database.getReference("Users").child(user_id)
    .child("ticketCount").setValue(
5                         ((MenuActivity)getActivity()).
    getCurrentUser().getTicketCount()+10
6                 );
7                 Toast.makeText(getActivity(), R.string.
    buy10Tickets, Toast.LENGTH_SHORT).show();
8             }
9         });
```

Listing 3.10: Metodo per l'acquisto di 10 biglietti.

3.7 HomeFragment

HomeFragment rappresenta la componente principale dell'applicazione. All'interno sono implementati 4 CardView che rappresentano le funzionalità

principali dell'applicazione: salvatare un posto di parcheggio, navigare al proprio veicolo, offrire e cercare posto di parcheggio. Le prime due funzionalità vengono implementate all'interno della componente, mentre le seconde in componenti diversi che verranno successivamente esaminati. Per quanto riguarda il salvataggio del posto di parcheggio, l'utente una volta che ha parcheggiato la macchina, cliccando sulla CardView Save, effettua il salvataggio della posizione attuale come posto di parcheggio del veicolo dell'utente, utilizzando i sensori di geolocalizzazione. La navigazione per il posto di parcheggio è correlata al salvataggio perché fa in modo che l'utente abbia una navigazione per raggiungere il proprio veicolo. Questa operazione viene effettuata attraverso un Intent che apre l'applicazione di Google Maps con destinazione le coordinate dell'ultimo parcheggio salvato. Un altro punto importante della componente è che prima di del reindirizzamento ai componenti di ricerca e di offerta di parcheggio, l'utente, attraverso un RecyclerView, visualizza la lista di automobili che ha precedentemente salvato e per procedere deve selezionare l'automobile che sta utilizzando premendo sull'elemento con le caratteristiche appropriate. In seguito la scelta viene salvata nel database. Riportiamo i metodi principali della componente:

```
1  private void setUpRecyclerView() {
2      final String user_id = mAuth.getCurrentUser().getUid();
3      Query query = FirebaseDatabase.getInstance().getReference
       ().child("Cars").child(user_id);
4      FirebaseRecyclerOptions<Car> options =
5              new FirebaseRecyclerOptions.Builder<Car>()
6                      .setQuery(query, Car.class)
7                      .build();
8      adapter = new CarAdapter(options);
9
10     mCarList.setAdapter(adapter);
11     adapter.setOnItemClickListener(new CarAdapter.
       OnItemClickListener() {
12     @Override
13     public void onItemClick(DataSnapshot documentSnapshot,
       int position) {
14         final String mCar= documentSnapshot.getKey();
15         new MaterialAlertDialogBuilder(getContext())
16         .setTitle(R.string.GetGiveConfirmationAlertTitle)
17         .setMessage(R.string.
       GetGiveConfirmationAlertSupportingText)
18         .setNegativeButton(R.string.AlertDialogDecline, new
       DialogInterface.OnClickListener() {
19             @Override
20             public void onClick(DialogInterface dialog, int
       which) {
21             }
```

```
22          }).setPositiveButton(R.string.AlertDialogAccept, new
    DialogInterface.OnClickListener() {
23              @Override
24              public void onClick(DialogInterface dialog, int
    which) {
25                  if(isGiveOperation&&!isGetOperation){
26                      giveParkingSpotFunction(mCar, user_id);
27                  }else if(!isGiveOperation&&isGetOperation){
28                      getParkingSpotFunction(mCar, user_id);
29                  }else{
30                      Toast.makeText(getContext(), "Operation
    failed",Toast.LENGTH_SHORT).show();
31                  }
32              }
33          }).setCancelable(false)
34          .show();
35
36      }
37      });
38  }
39  private void getLastKnownParkingSpot() {
40      final FirebaseDatabase database = FirebaseDatabase.
    getInstance();
41      DatabaseReference myRef = database.getReference("
    ParkingLocation").child(user_id);
42      myRef.addValueEventListener(new ValueEventListener() {
43          @Override
44          public void onDataChange(DataSnapshot dataSnapshot) {
45              lastKnownParkingSpot = dataSnapshot.getValue(
    GeoPoint.class);
46          }
47          @Override
48          public void onCancelled(DatabaseError databaseError)
    {
49              System.out.println("The read failed: " +
    databaseError.getCode());
50          }
51      });
52
53  }
54  private void navigateToParkingSpot() {
55      if (lastKnownParkingSpot != null) {
56          Uri gmmIntentUri = Uri.parse("google.navigation:q=" +
      lastKnownParkingSpot.getLatitude() + "," +
    lastKnownParkingSpot.getLongitude() + "&mode=w");
57          Intent mapIntent = new Intent(Intent.ACTION_VIEW,
    gmmIntentUri);
58          mapIntent.setPackage("com.google.android.apps.maps");
59          startActivity(mapIntent);
```

```
60      } else {
61          Toast.makeText(getContext(), R.string.
        unSuccessfulNavigationToParkingSpot, Toast.LENGTH_SHORT);
62      }
63  }
```

Listing 3.11: Metodi principali dell' HomeFragment.

3.8 DriverMapActivity

Questa attività rappresenta la funzionalità di ricerca di un posto di parcheggio da parte di un utente della rete. Come riportato nella fase di progettazione, l'assegnazione del posto di parcheggio viene effettuato ponendo priorità all'offerente. Quindi in questa attività non viene effettuata l'assegnazione, ma rimaniamo in atteso di un evento per il quale l'offerente nelle vicinanze effettua la proposta e il richiedente viene scelto come assegnatario del posto di parcheggio. In seguito portiamo il codice relativo:

```
1   private void getAssignedOfferer(){
2       String driverId = FirebaseAuth.getInstance().
        getCurrentUser().getUid();
3       DatabaseReference assignedOffererRef = FirebaseDatabase.
        getInstance().getReference().child("Users").child(driverId
        ).child("offerParking").child("customerRideId");
4       assignedOffererRef.addValueEventListener(new
        ValueEventListener() {
5           @Override
6           public void onDataChange(DataSnapshot dataSnapshot) {
7               if(dataSnapshot.exists()){
8                   status = 1;
9                   offererId = dataSnapshot.getValue().toString
        ();
10                  getAssignedParkingSpotLocation();
11                  getAssignedParkerInfo();
12                  getOfferCarInfo();
13                  getAssignedParkingSpotLocation();
14                  showAcceptDeclineButton();
15              }else{
16                  endRide();
17              }
18          }
19          @Override
20          public void onCancelled(DatabaseError databaseError)
        {
21          }
22      });
```

```
}
```

Listing 3.12: Attessa di assegnazione ad un posto di parcheggio.

Una volta che al richiedente li viene assegnato un posto di parcheggio, vengono recuperate tutte le informazioni relative all'altro utente e alla sua proposta: informazioni riguardanti il posto di parcheggio, l'automobile che sta utilizzando l'altro utente ecc.

```
 private void getAssignedParkingSpotLocation(){
    assignedParkingSpotLocationRef = FirebaseDatabase.
   getInstance().getReference().child("offerParking").child(
   offererId).child("l");
    assignedParkingSpotLocationRefListener =
   assignedParkingSpotLocationRef.addValueEventListener(new
   ValueEventListener() {
         @Override
         public void onDataChange(DataSnapshot dataSnapshot) {
             if(dataSnapshot.exists() && !offererId.equals("")
   ){
                 List<Object> map = (List<Object>)
   dataSnapshot.getValue();
                 double locationLat = 0;
                 double locationLng = 0;
                 if(map.get(0) != null){
                     locationLat = Double.parseDouble(map.get
   (0).toString());
                 }
                 if(map.get(1) != null){
                     locationLng = Double.parseDouble(map.get
   (1).toString());
                 }
                 parkingSpotLatLng = new LatLng(locationLat,
   locationLng);
                 pickupMarker = mMap.addMarker(new
   MarkerOptions().position(parkingSpotLatLng).title("Parked
   car"));
             }
         }

         @Override
         public void onCancelled(DatabaseError databaseError)
   {
         }
     });
}

private void getAssignedParkerInfo(){
    mOffererInfo.setVisibility(View.VISIBLE);
```

```
29      DatabaseReference mCustomerDatabase = FirebaseDatabase.
    getInstance().getReference().child("Users").child(
    offererId);
30      mCustomerDatabase.addListenerForSingleValueEvent(new
    ValueEventListener() {
31          @Override
32          public void onDataChange(DataSnapshot dataSnapshot) {
33              if(dataSnapshot.exists() && dataSnapshot.
    getChildrenCount()>0){
34                  Map<String, Object> map = (Map<String, Object
    >) dataSnapshot.getValue();
35                  if(map.get("name")!=null){
36                      mOffererName.setText(map.get("name").
    toString());
37                  }
38              }
39          }
40          @Override
41          public void onCancelled(DatabaseError databaseError)
    {
42          }
43      });
44  }
45  private void getOfferCarInfo(){
46      DatabaseReference mCustomerDatabase = FirebaseDatabase.
    getInstance().getReference().child("LastUsedCar").child(
    offererId);
47      mCustomerDatabase.addListenerForSingleValueEvent(new
    ValueEventListener() {
48          @Override
49          public void onDataChange(DataSnapshot dataSnapshot) {
50              String car_id = dataSnapshot.child("CarId").
    getValue().toString();
51              DatabaseReference mCustomerDatabase =
    FirebaseDatabase.getInstance().getReference().child("Cars"
    ).child(offererId).child(car_id);
52              mCustomerDatabase.addListenerForSingleValueEvent(
    new ValueEventListener() {
53                  @Override
54                  public void onDataChange(DataSnapshot
    dataSnapshot) {
55                      mOfferCarPlate.setText(dataSnapshot.child("
    plate").getValue().toString());
56                      mOfferCarBrand.setText(dataSnapshot.child
    ("brand").getValue().toString());
57                      mOfferCarModel.setText(dataSnapshot.child
    ("model").getValue().toString());
58                      mOfferCarColor.setText(dataSnapshot.child
    ("color").getValue().toString());
```

```
                    if(dataSnapshot.child("profileImageUrl").
    getValue()!=null){
                        Glide.with(getApplication()).load(
    dataSnapshot.child("carImage").getValue().toString()).into
    (mOffererProfileImage);
                    }
                }
                @Override
                public void onCancelled(DatabaseError
    databaseError) {
                }
            });
        }
        @Override
        public void onCancelled(DatabaseError databaseError)
    {
        }
     });
}
```

Listing 3.13: Prelievo dei dati dell'offerente.

Una volta prelevati tutti i dati necessari, l'utente, visualizzando sulla mappa la posizione del posto di parcheggio e la distanza dal punto attuale, decide se accettare o rifiutare la proposta. Nel caso la rifiuti, continua ad essere in attesa di altre proposte, nel caso in cui l'accetti, l'utente attraverso un Intent viene indirizzato alla navigazione di Google Maps con il posto di parcheggio da raggiungere e viene attivato un foreground service che tratteremo successivamente al punto 3.10.

```
private void showAcceptDeclineButton() {
    if(mAcceptParkingSpot.hasOnClickListeners()){
        mAcceptParkingSpot.setOnClickListener(null);
    }
    if(mDeclineParkingSpot.hasOnClickListeners()){
        mDeclineParkingSpot.setOnClickListener(null);
    }
    mAcceptParkingSpot.setOnClickListener(new View.
    OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent startServiceForeground = new Intent(v.
    getContext(),TrackingService.class);
            startService(startServiceForeground);
            Uri gmmIntentUri = Uri.parse("google.navigation:q
    =" + parkingSpotLatLng.latitude + "," + parkingSpotLatLng.
    longitude + "&mode=d");
            Intent mapIntent = new Intent(Intent.ACTION_VIEW,
     gmmIntentUri);
```

```
15              mapIntent.setPackage("com.google.android.apps.
    maps");
16              startActivity(mapIntent);
17          }
18      });
19      mDeclineParkingSpot.setOnClickListener(new View.
    OnClickListener() {
20          @Override
21          public void onClick(View v) {
22              String driverId = FirebaseAuth.getInstance().
    getCurrentUser().getUid();
23              DatabaseReference assignedCustomerRef =
    FirebaseDatabase.getInstance().getReference().child("Users
    ").child(driverId).child("offerParking");
24              assignedCustomerRef.removeValue();
25              mOffererInfo.setVisibility(View.INVISIBLE);
26          }
27      });
28  }
```

Listing 3.14: Metodo per l'accettazione o rifiuto del posto di parcheggio proposto.

3.9 OffererMapActivity

Questa componente rappresenta la funzionalità di proposta di un posto di parcheggio ad un altro utente richiedente. L'implementazione dell'assegnazione dell'utente in ricerca viene calcolata in base alla distanza minima tra gli utenti. Per effettuare tutto ciò è stato utilizzato Geofire, che effettua query basate sulla geolocalizzazione dell'utente. La lettura dei dati del richiedente è simile alla componente di prima, per questo motivo non viene riportata.

```
1  private int radius = 1;
2  private Boolean driverFound = false;
3  private String driverFoundID;
4  GeoQuery geoQuery;
5  private void getClosestDriver(){
6      DatabaseReference driverLocation = FirebaseDatabase.
    getInstance().getReference().child("driversAvailable");
7
8      GeoFire geoFire = new GeoFire(driverLocation);
9      geoQuery = geoFire.queryAtLocation(new GeoLocation(
    pickupLocation.latitude, pickupLocation.longitude), radius
    );
10     geoQuery.removeAllListeners();
11     geoQuery.addGeoQueryEventListener(new
    GeoQueryEventListener() {
```

```
12          @Override
13          public void onKeyEntered(String key, GeoLocation
    location) {
14              if (!driverFound && requestBol){
15                  DatabaseReference mCustomerDatabase =
    FirebaseDatabase.getInstance().getReference().child("Users
    ").child(key);
16                  mCustomerDatabase.
    addListenerForSingleValueEvent(new ValueEventListener() {
17                      @Override
18                      public void onDataChange(DataSnapshot
    dataSnapshot) {
19                          if (dataSnapshot.exists() &&
    dataSnapshot.getChildrenCount()>0){
20                              if (driverFound){
21                                  return;
22                              }
23                              driverFound = true;
24                              driverFoundID = dataSnapshot.
    getKey();
25                              DatabaseReference driverRef =
    FirebaseDatabase.getInstance().getReference().child("Users
    ").child(driverFoundID).child("offerParking");
26                              String customerId = FirebaseAuth.
    getInstance().getCurrentUser().getUid();
27                              HashMap map = new HashMap();
28                              map.put("customerRideId",
    customerId);
29                              driverRef.updateChildren(map);
30                              getDriverLocation();
31                              getDriverInfo();
32                              getDriverCarInfo();
33                              getHasRideEnded(); mRequest.
    setText(getString(R.string.looking_driver));
34                          }
35                      }
36                      @Override
37                      public void onCancelled(DatabaseError
    databaseError) {
38                      }
39                  });
40              }
41          }
42          @Override
43          public void onKeyExited(String key) {
44          }
45          @Override
46          public void onKeyMoved(String key, GeoLocation
    location) {
```

```
47          }
48          @Override
49          public void onGeoQueryReady() {
50              if (!driverFound)
51              {
52                  if(radius<300) {
53                      radius++;
54                  }
55                  getClosestDriver();
56              }
57          }
58          @Override
59          public void onGeoQueryError(DatabaseError error) {
60
61          }
62      });
63  }
```

Listing 3.15: Assegnazione dell'utente in cerca di un posto di parcheggio.

Dopo il recupero dei dati l'utente attende che il richiedente arrivi al posto di parcheggio prefissato ed effettui lo scambio. In seguito si attende che il richiedente confermi l'avvenuto scambio del posto di parcheggio.

```
1  private void getHasTransactionEnded(){
2      driveHasEndedRef = FirebaseDatabase.getInstance().
       getReference().child("Users").child(driverFoundID).child("
       offerParking").child("customerRideId");
3      driveHasEndedRefListener = driveHasEndedRef.
       addValueEventListener(new ValueEventListener() {
4          @Override
5          public void onDataChange(DataSnapshot dataSnapshot) {
6              if(dataSnapshot.exists()){
7
8              }else{
9                  endRide();
10             }
11         }
12         @Override
13         public void onCancelled(DatabaseError databaseError)
       {
14         }
15     });
16 }
```

Listing 3.16: Attesa che l'utente richeidente confermi lo scambio.

3.10 TrackingService

Si tratta di un foreground service il quale viene attivato nel momento in cui un utente in cerca di un posto di parcheggio accetta un offerta del sistema. In questo momento attraverso un Intent viene avviata la navigazione per il posto di parcheggio attraverso Google Maps, dato che non è possibile includerlo nell'applicazione, è stato creato questo foreground service con una notifica che contiene i dati relativi al veicolo dell'utente con cui si vuole effettuare lo scambio. Oltre alle informazioni relative al veicolo, nella notification ,attraverso due pending intents, viene data la possibilità di confermare che lo scambio è stato effettuato con successo o meno, in entrambi i casi viene effettuato lo scambio dei crediti.

```
1  private void buildNotification() {
2      Intent foundIntent = new Intent(this, AcceptParkingSpot.
       class);
3      PendingIntent pendingFoundIntent =
4              PendingIntent.getActivity(this, 0, foundIntent,
       0);
5      Intent notFoundIntent = new Intent(this,
       DeclineParkingSpot.class);
6      PendingIntent pendingNotFoundIntent =
7              PendingIntent.getActivity(this, 0, notFoundIntent
       , 0);
8      NotificationManager notificationManager = (
       NotificationManager) getSystemService(Context.
       NOTIFICATION_SERVICE);
9      Notification.Action found = new Notification.Action(R.
       drawable.yes, getString(R.string.found_parking_spot),
       pendingFoundIntent);
10     Notification.Action notFound = new Notification.Action(R.
       drawable.no, getString(R.string.not_found_parking_spot),
       pendingNotFoundIntent);
11     Notification notification =
12     new Notification.Builder(this, "Tracking Notification")
13         .setContentTitle(parkerCar.getPlate())
14         .setContentText(parkerCar.getBrand()+" "+parkerCar.
       getModel()+" "+parkerCar.getColor()+" "+" \n"+name)
15         .setContentIntent(pendingFoundIntent)
16         .setOngoing(true)
17         .addAction(found)
18         .addAction(notFound)
19         .build();
20     startForeground( 1,notification);
```

Listing 3.17: Notifica per la conferma dell'avvenuto scambio tra gli utenti.

Capitolo 4

Workflow dell'utilizzo

Quando l'utente apre l'applicazione per la prima volta, non viene individuato nessun profilo, per questo motivo dopo lo splash screen animato visualizza la pagina di log in. Nel caso in cui l'utente sia in possesso di credenziali, può effettuare l'accesso attraverso email e password, altrimenti dovrà effettuare la registrazione alla piattaforma. Una volta eseguito con succeso il Login, durante gli avvii successivi dell'applicazione, le credenziali non saranno più richieste e l'utente potrà accedere direttamente alle funzionalità della piattaforma fino ad un eventuale Logout. Riportiamo gli screen della pagina di Login e Registrazione:

Figura 4.1: Pagina di Login e registrazione.

Una volta effettuata l'autenticazione al sistema viene visualizzata la HomePage e scorrendo dal lato sinistro si potrà accedere al menù dell'applicazione come riportato di seguito:

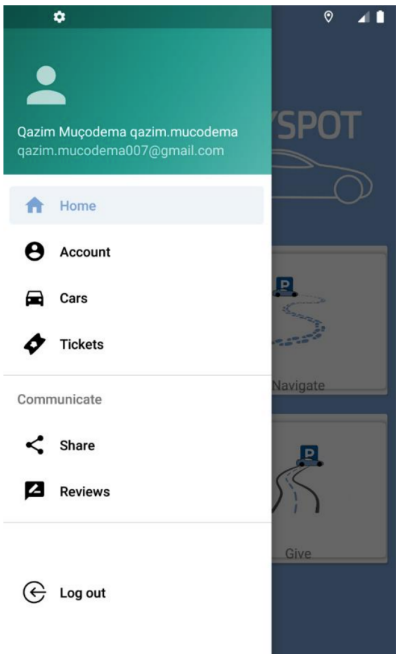

Figura 4.2: Menù dell'applicazione.

Attravero il menù l'utente è in grado di effettuare diverse operazioni che analizzeremo. Nel momento in cui dal menù viene selezionato Account, viene visualizzata questa schermata dove oltre a visualizzare i dati inseriti in fase di registrazione, è possibile sovrascriverli:

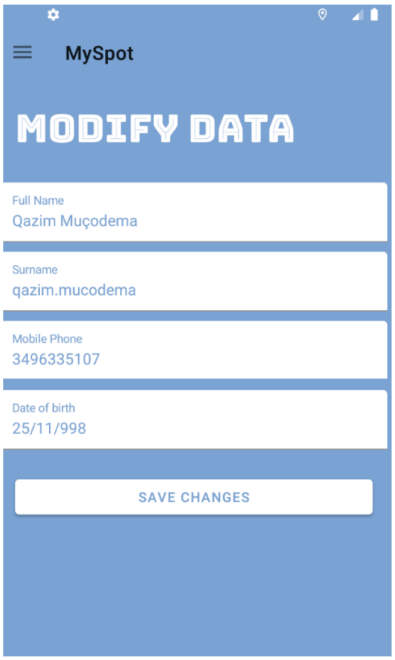

Figura 4.3: Pagina profilo.

Quando l'utente seleziona la sezione Cars, sarà in grado di visualizzare tutte le macchine che ha precedentemente salvato nella piattaforma. Attraverso lo swap, potrà decidere se eliminare una macchina che non possiede più e attraverso il button cliccabile, sarà in grado di aggiungere una nuovo veicolo visualizzando la seconda schermata dove potrà inserire tutti i dati relativi al nuovo mezzo di trasporto. Nella seconda schermata, cliccando sull'immagine della macchina, l'utente è in grado di effettuare l'upload di una foto che possiede in modo tale da renderla più riconoscibile.

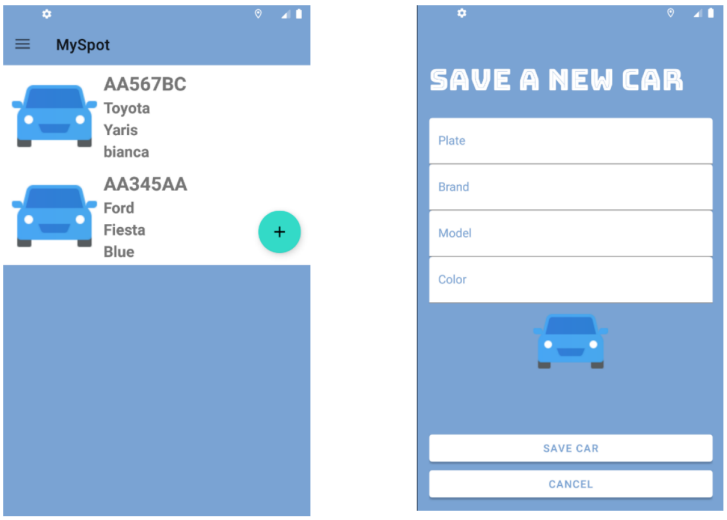

Figura 4.4: Pagina per la gestione dei veicoli.

Premendo sulla sezione tickets, l'utente potrà acquistare vari biglietti in base alle sue necessità. Le funzioni di effettivo pagamento non sono state implementate, al momento è possibile unicamente incrementare il numero di biglietti cliccando su uno dei pacchetti disponibili.

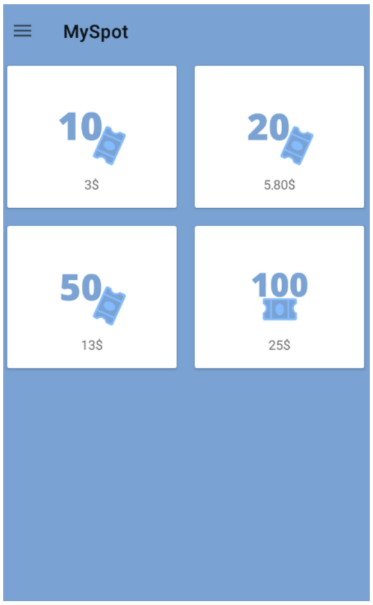

Figura 4.5: Pagina dei biglietti.

Le funzionalità principali dell'applicativo sono raggiungibili attraverso la pagina home, l'utente può:

- Salvare il posto di parcheggio.
- Navigare al proprio veicolo.
- Cercare un posto di parcheggio.
- Offrire un posto di parcheggio.

Figura 4.6: Pagina Home.

Quando l'utente preme su Save, viene chiesta una conferma per continuare l'operazion, al consenso viene salvata la posizione attuale identificata come il posto di parcheggio in cui la macchina è parcheggiata. Successivamente l'utente è in grado di risalire alla posizione del veicolo cliccando su Navigate, a questo punto viene iniziato il tracciamento dell'itinerario per il raggiungimento del veicolo attraverso l'applicazione di Google Maps, specifianco come modalità di movimeno l'opzione 'camminando'.

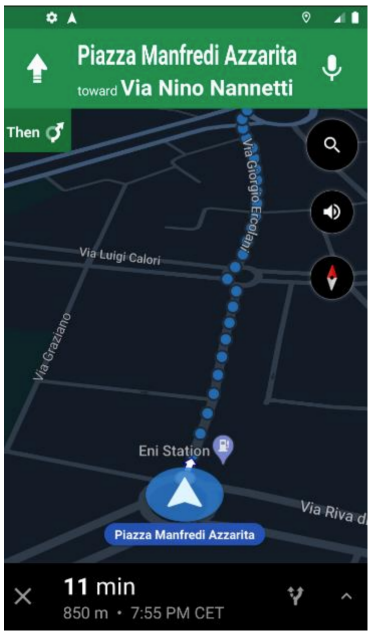

Figura 4.7: Navigazione al posto di parcheggio.

Se l'utente desidera cercare o offrire un posto di parcheggio, cliccando sulle Cards, viene reindirizzato su una pagina per selezionare il veicolo che sta utilizzando in modo tale da renderlo disponibile all'altro partecipante dello scambio.

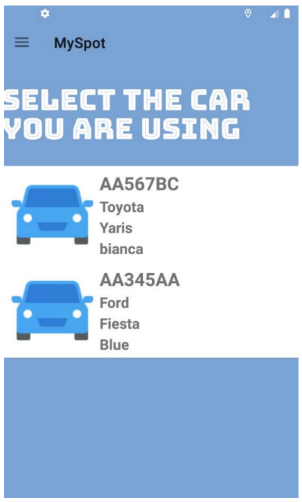

Figura 4.8: Pagina per la selezione del veicolo in uso.

Una volta completata la selezione del veicolo in uso, l'utente procede con la funzionalità che intende azionare. Se desidera offrire un posto di parcheggio viene reindirizzato ad una pagina che comprende una mappa nella quale, una volta individuato il richiedente, visualizza i suoi dati insieme alla posizione attuale in cui si trova come mostrato in seguito:

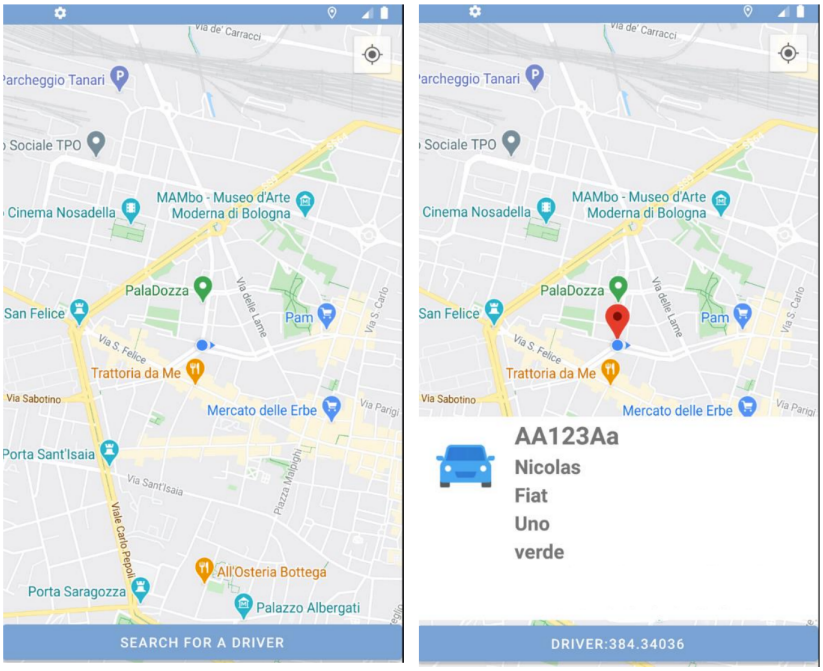

Figura 4.9: Offerta di parcheggio.

Nel caso invece in cui l'utente voglia cercare parcheggio, una volta assegnato un posto di parcheggio, può decidere se accettare o rifiutare in base alle proprie necessità e in seguito viene attivata la navigazione per raggiungere il parcheggio.

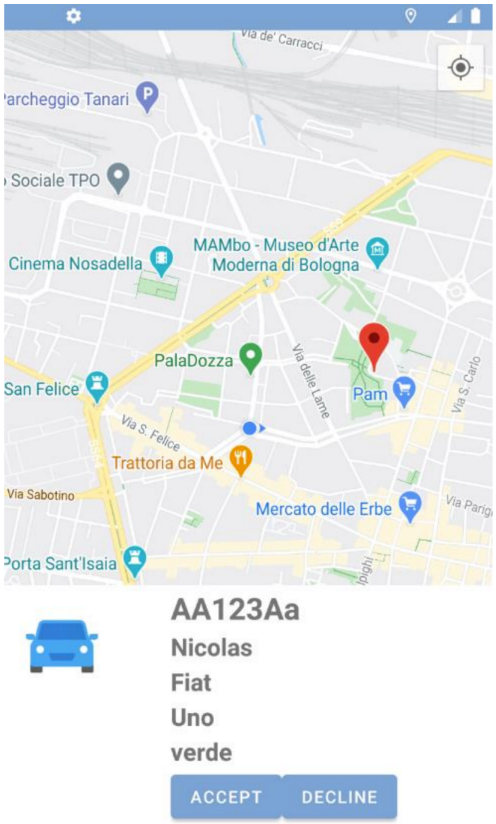

Figura 4.10: Richiesta di parcheggio.

Capitolo 5

Conclusioni

In questa tesi ho sviluppato un progetto che consente agli utenti di scambiare posti di parcheggio tra di loro, permutando biglietti che vengono utilizzati per effettuare le richieste e che sono acquistabili all'interno dell'applicazione. L'utilizzo di questo sistema di biglietti è innovativo per questo genere di applicazioni, come riportato nel capitolo dell'architettura, aiuta a stabilire l'equilibrio di Nash e fa in modo che l'offerente, nel caso in cui non rispetti lo scambio e non attendi il richiedente, abbia una perdita di biglietti. L'utilizzo massivo da parte degli utenti aiuta a diminuire il traffico in zone abitate, prodotto dalle persone in attiva ricerca per u posto di parcheggio. L'applicazione è stata sviluppata in Android per poter raggiungere più utenti possibili.

5.1 Stato attuale e limiti del progetto

Per quanto riguarda lo stato attuale del progetto, possiamo dire che è completamente funzionante, il posto di parcheggio dell'offerente viene comunicato al richiedente e la navigazione attraverso Google Maps, si riesce a raggiungere la destinazione. Sono state implementate le basi per la valutazione degli utenti in base alla correttezza, si pensa di introdurre alla conclusione dello scambio una valutazione da 0 a 5 stelle. Al momento l'applicativo non è pubblicato in nessun servizio di distribuzione digitale, in quanto non è stata testata la funzinalità a larga scala per migliaia di utenti. Prima della pubblicazione, inoltre, bisogna effettuare uno studio sulla legalità di questo sistema, controllare se sia possibile in ambito giuridico trasferire il posto di parcheggio di proprietà comunale ad un altra persona.

5.2 Sviluppi futuri

Stabilire un prezzo valido per i biglietti si è rilevato un problema non banale, per risolverlo penso sia possibile effettuare uno studio di mercato per definire quante liquidità monetarie le persone siano disposte a pagare per la sicurezza di trovare un parcheggio, un altra implementazione futura potrebbe essere quella di far decidere all'offerente quanti biglietti chiedere ai vari utenti per il parcheggio, l'applicazione in quel momento potrebbe stimare e consigliare una quantità adeguata in base alla posizione e all'orario in cui viene proposto il parcheggio. Nel futuro sarebbe opportuno effettuare una simulazione per capire tutti i punti deboli e per poter effettuare un tuning e garantire una performanza ottima agli utenti. Sarebbe opportuno inoltre sviluppare l'applicativo per il sistema operativo IOS, in modo tale da raggiungere più utenti. In un futuro vorrei implementare dei controlli riguardanti l'avvenimento dello scambio. Per quanto riguarda l'offerente, ho pensato a calcolare il tempo necessario che l'utente impiega all'arrivo valutando fattori come il traffico in tempo reale, per poi far partire un timer allo scadere del quale l'offerente potrà lasciare il posto di parcheggio e lo scambio verrà considerato come un successo. Per quanto riguarda l'offerente invece vorrei implementare un controllo attraverso "l'activity recognition", per cui nel caso in cui l'utente affermi che lo scambio non è stato effettuato, venga controllato attraverso l'analisi del contesto se stia ancora guidando o se stia mentendo. Ritengo che operando in questo modo l'applicativo potrà avere maggiori possibilità di successo alla pubblicazione.

Ringraziamenti

Inanzitutto ringrazio la mia famiglia per avermi sostenuto e incentivato a intraprendere questo percorso formativo che rappresenta un periodo molto importante nella mia vita. Ci sono stati molti alti e bassi, ho avuto la fortuna di conoscere molte persone e di stringere nuove amicizie, ringrazio tutti i miei colleghi universitari per avere lavorato con me e mi hanno aiutato a portare a termine la mia carriera universitaria. Un particolare ringraziamento a tutti gli amici di Bologna che mi hanno fatto sentire a casa anche se a chilometri di distanza dall'Albania. Infine ringrazio il Dott. Federico Montori per aver creduto nel progetto e avermi sostenuto a portarlo a termine, è sempre stato disponibile e comprensibile nei miei confronti.

Bibliografia

[1] Parqex app website. `https://www.parqex.com/`.

[2] Gregory D Abowd, Anind K Dey, Peter J Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *International symposium on handheld and ubiquitous computing*, pages 304–307. Springer, 1999.

[3] M Hadi Amini and Orkun Karabasoglu. Optimal operation of interdependent power systems and electrified transportation networks. *Energies*, 11(1):196, 2018.

[4] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[5] Ana M Bernardos, Paula Tarrio, and Jose R Casar. A data fusion framework for context-aware mobile services. In *2008 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 606–613. IEEE, 2008.

[6] David L Brock. The electronic product code (epc). *Auto-ID Center White Paper MIT-AUTOID-WH-002*, pages 1–21, 2001.

[7] Felix Caicedo. Real-time parking information management to reduce search time, vehicle displacement and emissions. *Transportation Research Part D: Transport and Environment*, 15(4):228–234, 2010.

[8] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. 2000.

[9] Shuo-Yan Chou, Shih-Wei Lin, and Chien-Chang Li. Dynamic parking negotiation and guidance using an agent-based platform. *Expert Systems with applications*, 35(3):805–817, 2008.

[10] Andrea R Demegillo, John Kerwin D Dizon, Kenneth Bryan A Talon, Francis F Balahadia, and Orlando M Lingo. Real-time viewing automated parking system. In *2016 IEEE Region 10 Symposium (TENSYMP)*, pages 145–149. IEEE, 2016.

[11] Mathias Gabriel Diaz Ogás, Ramon Fabregat, and Silvana Aciar. Survey of smart parking systems. *Applied Sciences*, 10(11):3872, 2020.

[12] Angelika Dohr, Robert Modre-Opsrian, Mario Drobics, Dieter Hayn, and Günter Schreier. The internet of things for ambient assisted living. In *2010 seventh international conference on information technology: new generations*, pages 804–809. Ieee, 2010.

[13] K. Hassoune, W. Dachry, F. Moutaouakkil, and H. Medromi. Smart parking systems: A survey. In *2016 11th International Conference on Intelligent Systems: Theories and Applications (SITA)*, pages 1–6, 2016.

[14] DG INFSO et al. Internet of things in 2020: Roadmap for the future. *INFSO D*, 4, 2008.

[15] Sam Judd. Glide documentation. `https://github.com/bumptech/glide`.

[16] Gerd Kortuem, Fahim Kawsar, Vasughi Sundramoorthy, and Daniel Fitton. Smart objects as building blocks for the internet of things. *IEEE Internet Computing*, 14(1):44–51, 2009.

[17] Danh Le-Phuoc, Axel Polleres, Manfred Hauswirth, Giovanni Tummarello, and Christian Morbidoni. Rapid prototyping of semantic mash-ups through semantic web pipes. In *Proceedings of the 18th international conference on World wide web*, pages 581–590, 2009.

[18] Ying Chen Lin and Jung Hong Hong. The combination of the parking spaces emergence response and iot technology—using flood as an example. In *Proceedings of the ACRS*. Citeseer, 2015.

[19] Luca Mainetti, Luigi Patrono, Maria Laura Stefanizzi, and Roberto Vergallo. A smart parking system based on iot protocols and emerging enabling technologies. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 764–769. IEEE, 2015.

[20] David Martín, Carlos Lamsfus, and Aurkene Alzua. Automatic context data life cycle management framework. In *5th International Conference on Pervasive Computing and Applications*, pages 330–335. IEEE, 2010.

[21] Sujith Samuel Mathew, Yacine Atif, Quan Z Sheng, and Zakaria Maamar. Building sustainable parking lots with the web of things. *Personal and ubiquitous computing*, 18(4):895–907, 2014.

[22] R Mercuri, A Bauen, and D Hart. Options for refuelling hydrogen fuel cell vehicles in italy. *Journal of power sources*, 106(1-2):353–363, 2002.

[23] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Context aware computing for the internet of things: A survey. *IEEE Communications Surveys Tutorials*, 16(1):414–454, 2014.

[24] Fano Ramparany, Remco Poortinga, Maja Stikic, Jorg Schmalenstroer, and Thorsten Prante. An open context information management infrastructure-the ist-amigo project. 2007.

[25] Luis Sanchez, Jorge Lanza, Rasmus Olsen, Martin Bauer, and Marc Girod-Genet. A generic context management framework for personal networking environments. In *2006 Third Annual International Conference on Mobile and Ubiquitous Systems: Networking & Services*, pages 1–8. IEEE, 2006.

[26] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *1994 First Workshop on Mobile Computing Systems and Applications*, pages 85–90. IEEE, 1994.

[27] Cui Shiyao, Wu Ming, Liu Chen, and Rong Na. The research and implement of the intelligent parking reservation management system based on zigbee technology. In *2014 Sixth International Conference on Measuring Technology and Mechatronics Automation*, pages 741–744. IEEE, 2014.

[28] Harald Sundmaeker, Patrick Guillemin, Peter Friess, and Sylvie Woelfflé. Vision and challenges for realising the internet of things. *Cluster of European Research Projects on the Internet of Things, European Commision*, 3(3):34–36, 2010.

[29] Lu Tan and Neng Wang. Future internet: The internet of things. In *2010 3rd international conference on advanced computer theory and engineering (ICACTE)*, volume 5, pages V5–376. IEEE, 2010.

[30] Righa Tandon and PK Gupta. Optimizing smart parking system by using fog computing. In *International Conference on Advances in Computing and Data Sciences*, pages 724–737. Springer, 2019.

[31] Ovidiu Vermesan, Peter Friess, Patrick Guillemin, Sergio Gusmeroli, Harald Sundmaeker, Alessandro Bassi, Ignacio Soler Jubert, Margaretha Mazura, Mark Harrison, Markus Eisenhauer, et al. Internet of things strategic research roadmap. *Internet of things-global technological and societal trends*, 1(2011):9–52, 2011.

[32] Jihoon Yang, Jorge Portilla, and Teresa Riesgo. Smart parking service based on wireless sensor networks. In *IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society*, pages 6029–6034. IEEE, 2012.

[33] Arkady Zaslavsky, Charith Perera, and Dimitrios Georgakopoulos. Sensing as a service and big data. *arXiv preprint arXiv:1301.0159*, 2013.