POLITECNICO DI TORINO

SEDE DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Linguaggi multipiattaforma per sviluppo di applicazioni mobile in ambienti operativi di waste management

Relatore
prof. Alfredo Benso

Candidato
Emanuele Raineri
matricola: 241881

Anno accademico 2019-2020

Sommario

La tesi tratta il framework React Native, elencando in via teorica il suo funzionamento e i vari vantaggi e svantaggi. Vengono inoltre trattati i pro e contro del suo utilizzo a confronto con framework di linguaggi mobile che utilizzano un approccio simile. Successivamente nella tesi vengono esposti due applicativi sviluppati entrambi in React Native e le best-practice assimilate nello svolgimento di essi.

Ringraziamenti

Vorrei ringraziare il professor Alfredo Benso, Guglielmo Savino e tutti i componenti di Moltosenso Srl per avermi supportato nel periodo di stage e nella stesura di questa tesi fornendomi le consocenze con la professionalità che li distingue. Ringrazio inoltre la mia famiglia che mi ha permesso di arrivare a questo punto, e tutte le persone che ho incontrato nel mio percorso di studi e che mi hanno supportato.

Indice

Elenco delle figure

Introduzione

L'obbiettivo principale della tesi consiste nell'imparare a sviluppare e conoscere un "nuovo" linguaggio di programmazione in ambiente mobile chiamato React-Native. Essa è una libreria open source, utilizzabile per lo sviluppo di applicazioni mobile cross-platform (usufruibile sui diversi sistemi operativi quali Android, iOS, windows, ecc.) che utilizza un approccio pseudo-nativo. E' una libreria rilasciata da Facebook nel 2015 ed è completamente basata sul linguaggio React JS, che a sua volta è una libreria JavaScript utilizzata per la creazione di interfacce utente.

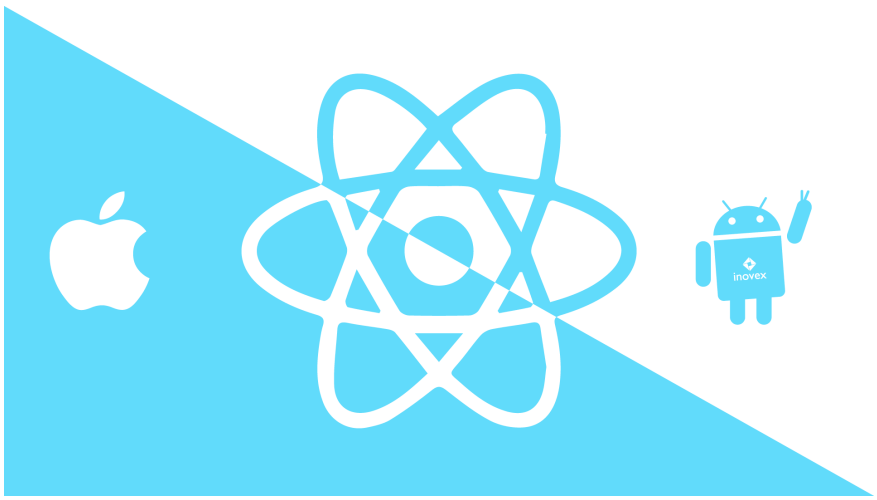

Figura 1: Immagine rappresentativa di React-Native, la quale prende il simbolo centrale, l'atomo, come eredità dal linguaggio dal quale prende spunto, React JS, e inoltre i differenti colori rappresentano il cross platform

Gli obbiettivi principali della tesi sono due: approfondire un linguaggio innovativo e multipiattaforma tramite una parte di ricerca per

poter applicare i concetti imparati direttamente implementando due applicativi.

La parte implementativa consiste nello sviluppare due applicativi in React Native in collaborazione con MoltoSenso s.r.l. e CIDIU S.p.a.. Il primo di essi è una applicazione , chiamata Tag Reader App, che ha come scopo la comunicazione tra un hardware dedicato (lettore di tag) e un database online, con una manipolazione intermedia dei dati da parte dell'utente. La realizzazione è tesa a semplificare e migliorare il processo di censimento dei contenitori dei rifiuti per CIDIU S.p.a.. La seconda applicazione, è un refactoring in chiave React Native di una precedente app Android nativa, già presente sullo store dedicato. L'applicazione, nello specifico, si chiama CIDIU4U ed è l'app ufficiale di CIDIU S.p.a., dedicata all'utenza cittadina del servizio di raccolta rifiuti dei Comuni serviti da CIDIU e già raggiunti dalla tecnologia O.N.D.E-UWC[1]. Con questa app è possibile verificare la posizione dei mezzi di raccolta in tempo reale, segnalare a Cidiu un disservizio sul territorio, consultare la posizione dei siti di raccolta, monitorare il costo e la produttività di ogni raccoglitore stradale, scoprire dove viene smaltita ogni frazione di materiale conferita dagli utenti, e come ultima cosa, per gli utenti dotati di badge, è possibile prendere visione della lista degli ultimi accessi e giorno di conferimento. Oltre alle funzioni già esistenti, in questo nuovo progetto, si sono aggiunte alcune funzionalità come il calendario della raccolta rifiuti o la possibilità di poter filtrare alcune posizioni specifiche di contenitori sulla mappa. Bisogna inoltre tenere conto che questa app non è un progetto, ma un prodotto fatto e finito, dato che è già presente sul mercato, e perciò dovrà sottostare a determinati vincoli e regole, differenti dalla creazione di un progetto completamente scolastico.

La parte di analisi si concentra propriamente sul linguaggio di React Native, sul funzionamento, architettura, definizione e best-practice da utilizzare per la stesura di un prodotto che sia capibile da tutti e che sia in grado di essere modificato e/o aggiornato da qualunque sviluppatore terzo in qualsiasi momento. Un ulteriore sguardo viene dato alle differenze con i linguaggi con tecnologie "simili" ovvero linguaggi

[1]E' un progetto che ha come obbiettivo ottimizzare i costi garantendo comunque efficacia all'interno del sistema

ibridi/web/nativi e alle differenze con altri linguaggi ma con la stessa tecnologia pseudo-nativa per poter evidenziare i vantaggi e svantaggi dell'uso di React Native.

La tesi si organizza in cinque capitoli, il primo finalizzato all'introduzione per React Native nel quale sarà approfondito il funzionamento e analizzata nel dettaglio la natura del framework di sviluppo. La seconda sezione approfondisce i motivi della scelta di React Native come strumento di sviluppo, ponendo l'accento sulle differenze con altri linguaggi di programmazione sia nativi che web oriented e, di conseguenza, sui pro e contro delle varie soluzioni. Nel terzo capitolo è presente la spiegazione del progetto relativo al Tag Reader App, con una descrizione in generale e successivamente più nei dettagli, come si è svolta l'applicazione in sé, quali caratteristiche, activity e nello specifico quali soluzioni sono state utilizzate. Il quarto capitolo introduce a sua volta il secondo applicativo, CIDIU4U, utilizzando gli stessi canoni del capitolo precedente. L'ultimo capitolo si preoccupa di spiegare e descrivere le best practices utilizzate nella stesura delle due applicazioni per permettere un utilizzo migliore di React Native.

React-Native

Descrizione del funzionamento di React Native e spiegazione delle varie differenze con altri linguaggi pseudo simili

Capitolo 1

React Native: concetti principali

React Native è un linguaggio nuovo e innovativo ed ha una popolarità in continua crescita, basti pensare che il primo rilascio della libreria su GitHub è stato il 26 marzo 2015 e ad oggi si presenta una community con decine di migliaia di programmatori. L'ideatore di questo linguaggio è un colosso come Facebook, il che rassicura molti utenti, dato che è sempre in continuo aggiornamento e si trascina al suo fianco un'importante community di sviluppatori fidati.

1.1 Significato del nome

React Native è un nome composto da "React" e "Native", il primo è dovuto alla libreria ReactJS, da cui React Native prende maggiore spunto. React è una libreria Javascript open source ideata per la creazione di interfacce utente sul web. E' stata sviluppata nel 2011 e poi successivamente ripresa ed estesa da Facebook, grazie al quale è riuscita a guadagnare popolarità. Le caratteristiche principali di questa libreria sono l'adattabilità, la facilità d'uso e le alte performance che si riflettono anche su React Native dato che si basa proprio su di essa per la creazione dei componenti e specialmente della UI.

Tramite la parole "Native", si evidenzia il fatto che con React Native non si costruiscono applicazioni ibride o web, sia in Android che iOS o

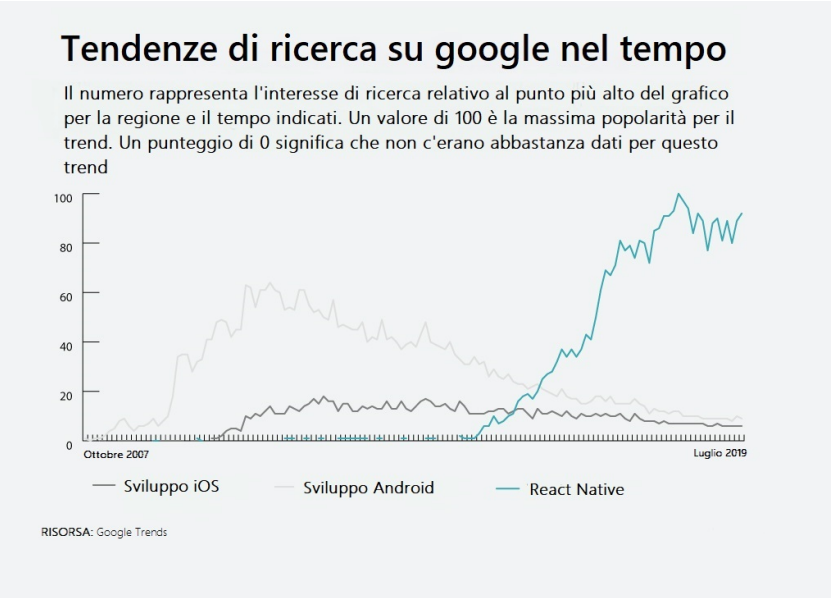

Figura 1.1: Popolarità di ricerca della parola React Native nel tempo

qualunque sistema operativo, ma si sviluppano le applicazioni quasi a livello nativo. React Native offre una funzione di "traduzione" che esegue il codice sviluppato da React Native (principalmente Javascript) in Java per Android e Objective-C per iOS. L'utilizzo di applicazioni native è un grosso vantaggio poichè migliorano l'applicazione in termini di velocità, affidabilità, risoluzione, accesso alle risorse, utilizzo offline, notifiche, ecc. Con React Native, che utilizza un approccio pseudo-nativo, oltre ad acquisire tutti questi vantaggi di un'applicazione nativa, acquisisce l'unico effettivo vantaggio dell'utilizzo di un'applicazione ibride, ovvero che sono app multipiattaforma. L'approccio pseudo-nativo permette perciò di avere quasi un'identica user experience e delle perfomance quasi alla pari delle applicazioni completamente native.

1.2 Funzionamento

React Native è un framework per lo sviluppo di applicazioni su dispositivi mobile con un approccio pseudo-nativo (simile al linguaggio propriamente nativo ma con un deterioramento delle prestazioni dovuto alla traduzione del linguaggio). Si basa principalmente su ReactJS, ovvero una libreria open source Javascript la quale è fondata sul linguaggio JSX [1]. Tramite la collaborazione tra JSX e Javascript in React, si producono i risultati della GUI (Graphical User Interface) andando a descrivere come ogni componente dovrebbe essere, dimensioni, posizionamento, ecc. e creando così l'interfaccia utente. React, e di conseguenza anche React Native, si basa su un concetto principale, il Componente, ogni elemento presente in un'applicazione (Button, View, Dialog, Text, Image, ecc.) è riconducibile a un componente. Esso è il singolo elemento che contribuisce alla creazione della logica e dell'interfaccia grafica ed è qualcosa che ha uno stato, può accettare delle proprietà, ed ha un suo ciclo di vita.

Il componente solitamente è utilizzato per creare altri componenti più complessi partendo da uno di base più semplice, "atomico", questo è dovuto alla loro proprietà di essere componibili. I React components sono visti come funzioni pure e parti integranti dell'interfaccia grafica, infatti l'interfaccia viene espressa tramite l'utilizzo del metodo render() che può a sua volta accedere allo stato corrente e alle proprietà. Questi componenti, sono reali e vengono messi a disposizione dal sistema operativo del dispositivo sul quale è avviata l'applicazione, e l'obbiettivo di React Native è quello di utilizzare Javascript e React per posizionarli e definirne la logica.

I componenti stanno alla base del funzionamento e della logica di React Native e contribuiscono a fornire informazioni e istruzioni all'interfaccia grafica incaricata di rappresentarle.

All'interno delle applicazioni mobile, per la visualizzazione delle informazioni, si utilizza il DOM ("Document Object Model", API che

[1] Estensione sintattica di Javascript che permette di scrivere la struttura dei vari componenti usando una sintassi simile al linguaggio HTML.

permette la rappresentazione di documenti HTML e XML[2]), esso rappresenta l'interfaccia utente dell'applicazione e ogni volta che si verifica una modifica nello stato o nelle informazioni contenute nella logica, il DOM viene aggiornato per rappresentare tale modifica. Le frequenti manipolazioni all'interfaccia grafica dovute alle modifiche apportate, possono portare a un rallentamento delle prestazioni. La modifica è un processo veloce, grazie alla struttura ad albero del DOM, ma il re-rendering anche delle parti non interessate a modifica crea un deterioramento delle prestazioni. Come soluzione al problema React native ha introdotto il concetto di Virtual DOM.

1.2.1 Virtual DOM

Il Virtual DOM è un'astrazione, una rappresentazione virtuale copiata in memoria del DOM reale. Anche in questo caso, come nel DOM reale, il Virtual DOM è rappresentato tramite uno schema ad albero dove ogni elemento è rappresentato da un nodo, di modo da facilitarne la modifica. Ogni qual volta che cambia lo stato dell'applicazione in questione viene aggiornato solamente il DOM virtuale, creando un nuovo albero ma mantenendo salvato in memoria il precedente. Dopo la creazione di un nuovo albero, esso viene confrontato con l'albero del DOM virtuale prima dei cambiamenti di stato, e viene fatta un'analisi delle differenze[3]. Le differenze trovate sono le effettive modifiche da apportare al nuovo rendering dell'applicazione. Sapendo le esatte modifiche da apportare al DOM reale, è più semplice trovare il metodo migliore per svolgere le operazioni richieste e inoltre non si effettua più un re-rendering di componenti rimasti tali e quali anche dopo il cambiamento di stato. Un aggiornamento mirato produrrà una riduzione delle risorse impiegate e una diminuzione del tempo di aggiornamento, e quindi benefici in termini di prestazione per l'app. La gestione delle differenze e tutti i dettagli inerenti ad esso sono completamente trasparenti per l'utente dato che React si occupa di questo confronto.

[2]Definisce la struttura logica di un documento e la via per accedere e manipolare quel documento

[3]Questo processo viene chiamato "diffing"

Come si può vedere nella figura sottostante, le modifiche da apportare sono evidenziate dal colore verde e rosso, mentre nel DOM reale iniziale i nodi non sottoposti a modifiche sono blu, utilizzando il Virtual DOM i nodi in blu rimangono tali e quali e non necessitano di un superfluo re-rendering.

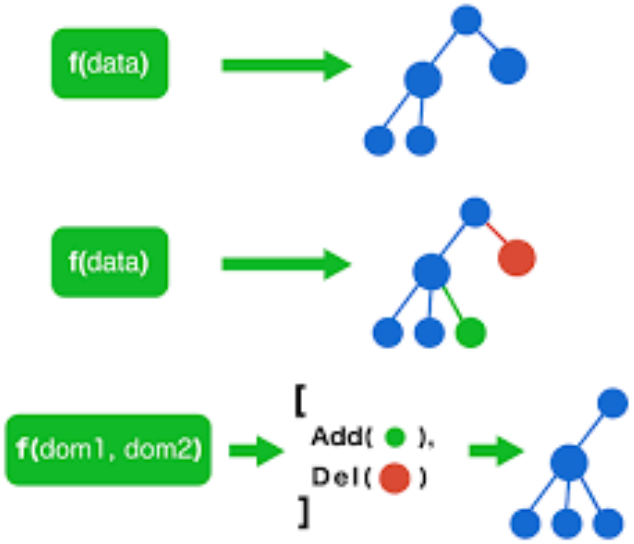

Figura 1.2: Esempio di funzionamento del virtual DOM

1.2.2 State e props

Un generico componente possiede due caratterizzazioni specifiche: State e Props. Entrambi gli elementi controllano la vita del componente. Le Props sono proprietà costanti del componente, mentre lo State gestisce ed indica la notifica degli eventi occorsi.

Le Props sono intese come diminutivo di proprietà ed hanno una regola principale, sono immutabili. Essendo immutabili sono variabili read-only, e perciò una volta ricevute in input non potranno mai essere modificate da parte di quel componente, devono rimanere pure. Ogni componente di React necessita di alcune props per la sua inizializzazione ed è per questo che esse si possono muovere in un sola direzione, dal componente padre al componente figlio e non viceversa. Le props hanno perciò due fondamentali funzionamenti: il primo consiste nel creare una

relazione ulteriore tra componente padre e componente figlio, mentre il secondo è la personalizzazione del componente. Stessi componenti con props differenti hanno un comportamento diverso.

Gli State hanno un comportamento totalmente differente dalle Props. Essi sono stati creati appositamente per essere mutevoli nel tempo all'interno di un componente. Gli State vengono utilizzati dai componenti per tenere traccia delle loro informazioni e gestire le interazioni con l'utente. Ogni qual volta che un componente ha bisogno di gestire dei dati mutevoli nel tempo, si utilizzano gli State. Se è presente un'interazione da parte dell'app con l'utente, ogni qual volta che l'utente preme un bottone o inserisce del testo, essa viene gestita dagli State presenti. Gli State assumeranno un valore differente a seconda della logica implementata per una determinata interazione. Per la modifica di uno state non è possibile accedervi direttamente, questo perchè React mette a disposizione una funzione chiamata setState() per l'aggiornamento di essi. La funzione *setState* risulta essere vantaggiosa in primo luogo perchè gestisce automaticamente gli eventi di modifica a differenza di altri linguaggi che necessitano di alcuni handler per essere usati correttamente. Oltre a una questione di sicurezza e gestione, la chiamata alla funzione *setState* crea nuovamente un re-rendering del componente e di tutti i suoi componenti figlio di modo da poter attuare le modifiche appena effettuate. Quest'ultima azione è opzionale, dato che tramite l'uso della funzione *shouldComponentUpdate()* è possibile evitare un re-rendering del componente.

Per concludere, l'unione di tutti questi elementi, proprietà, componenti, virtual DOM, state, props, unito anche a CSS (fogli di stile, utili per determinare lo stile degli elementi), serviranno a completare la nostra app in React Native.

1.3 Generi di applicazione per il quale è adatto e non

Essendo un linguaggio nuovo, React Native ha sia punti di forza che difetti e analizzandoli risulta essere un linguaggio più adatto ad alcune tipologie di applicazioni, mentre meno per altre che richiedono ulteriori o differenti specifiche del linguaggio. I vari pro e contro che si creano utilizzando React Native sono elencati nella sezione successiva.

L'uso di React Native è perciò fortemente consigliato per alcune categorie di applicazioni:

1. *E-Commerce.* La maggior parte degli utenti mobile trascorrono un gran parte del loro tempo ad utilizzare i propri dispositivi elettronici, specialmente utilizzando applicazioni di E-commerce come (Amazon, Subito, Ebay, ecc.) React Native si adatta perfettamente a questa tipologia dato che riesce a soddisfare alcuni requisiti che questo genere di applicazioni richiedono come un'interfaccia intuitiva e veloce, ottimizzazione del tempo dell'utente e affidabilità. Grazie a React Native e alla sua proprietà di linguaggio multiapiattaforma, i punti di e-commerce, possono ridurre al minimo aggiornamenti rapidi e mantere una coerenza del design anche su dispositivi differenti e sistemi operativi differenti.

2. *Social Media.* React Native è nato grazie al colosso Facebook, il quale insieme ad Instagram utilizza tutt'oggi React Native come linguaggio per le loro applicazioni disponibili sugli store. Le applicazioni sviluppate dei social media, sono app principalmente dirottate sulla visualizzazione di dati con caratteristiche principali come un'ottima user experience, scalabilità e velocità. Tramite React Native è possibile soddisfare tutte queste caratteristiche facendo risultare RN un linguaggio di programmazione adatto per questo tipo di applicazioni. Essendo cross-platform per applicazioni con un grosso seguito, esso pone fine anche alla rivalità fra vari dispositivi e sistemi operativi differenti dato che le versioni differenti dell'applicazione usciranno senza ritardi fra loro.

3. *Applicazioni Javascript.* Javascript è uno dei linguaggi più comuni a tutti i programmatori dato che è uno dei primi linguaggi browser

web ad essere uscito. E' inoltre anche uno dei più utilizzati grazie alla sua facilità di apprendimento e semplicità di implementazione tramite una elaborazione lato client, si tratta quindi di un linguaggio potente e con una grande community al seguito. Grazie alla facilità di scrittura, Javascript è largamente usato per applicazioni con poca elaborazione (rappresentano la maggior parte delle applicazioni disponibili sul mercato) e React Native avendo origine da ReactJS che a sua volta è basato su javascript, risulta il framework ideale per trarre vantaggio da una base di Javascript e migliorare lo sviluppo dell'applicazione mobile.

4. *Integrato con codice nativo.* Una proprietà di React Native è che può essere integrato con altri codici completamente nativi, perciò uno sviluppatore può considerare il fatto di sviluppare le funzioni principali per la propria applicazione in linguaggio nativo (Java, Kotlin, Objective-C, ecc.) per sfruttare al massimo le componenti native del dispositivo sul quale verrà fatta funzionare l'applicazione. Utilizzando React native, integrato, se ne possono trarre i vantaggi anche solo sulle singole activity o pagine come ad esempio activity delle informazioni, Faq, contatti, ecc. Così facendo le applicazioni non saranno efficienti in termini di costo del lavoro come applicazioni completamente in React Native, ma avremo una diminuzione dei tempi e dei costi per le parti sviluppate in comune e prestazioni migliori per le parti completamente in nativo.

5. *Velocità di pubblicazione.* Uno sviluppatore, nella maggior parte dei casi, incorre nella necessità di pubblicare la propria applicazione in tempi brevi sul mercato e quindi con versioni stabili per tutti i dispositivi e sistemi operativi. In queto caso utilizzare il framework React Native è fortemente consigliato dato che andrebbe a eliminare i ritardi dovuti allo sviluppo delle differenze tra versioni, limitando al minimo le modiche fra di essi. Inoltre grazie alla proprietà dell' "hot reloading", React Native permette di eseguire modifiche live senza ricompilazione e velocizzando il processo di testing e debug. E' perciò consigliato per applicizioni che richiedono aggiornamenti frequenti e brevi interazioni.

Oltre ad esserci casi di applicazioni per cui l'uso di React-Native è consigliato ci sono casi anche per cui l'uso di React Native è invece sconsigliato.

1. *Giochi e visualizzazioni complesse.*React Native utilizza un approccio pseudo-nativo, il che vuol dire che si avvicina molto a un'applicazione nativa ma ha comunque prestazioni peggiori. I giochi richiedono una forte interazione con l'utente e un alto livello di personalizzazione e perciò avranno prestazioni con performance migliori sicuramente framework interamente nativi dato che riescono a sopportare progetti complessi e ricchi di funzionalità. Tuttavia React Native sti sta comunque muovendo in questo campo cercando di aggiornarsi per raggiungere le controparti native, un esempio sono lo sviluppo di applicazioni come Discovery VR, app con funzionalità molto complesse come visualizzatore 3D.

2. *App specifiche per una piattaforma.* Se l'intenzione di uno sviluppatore è quella di scrivere un'applicazione interamente per un unico sistema operativo, l'utilizzo di React Native è fortemente sconsigliato. Uno dei maggiori vantaggi di React Native è l'essere un linguaggio multipiattaforma, se in questo caso esso non è un vantaggio allora è consigliabile continuare a programmare con un linguaggio nativo di quel sistema operativo di modo da massimizzare le prestazioni. In ogni caso comunque le applicazioni sviluppate unicamente per una sola piattaforma sono molto rare.

3. *Applicazione sempre attive.* Molte applicazioni multimediali o di messaggistica (quindi istantanee) hanno una moltitudine di processi in esecuzione in background. React Native è ancora agli inizi nel campo dei processi e si sta portando avanti cercando di migliorare il threading model anche grazie al continuo investimento di Facebook in questa tecnologia e nella comunità di supporto. Per app di piccole dimensioni il problema non sussiste perchè la perdita di prestazioni sarà minima, mentre per le applicazioni con molti threads attivi in background si può incorrere in un collo di bottiglia. In un futuro prossimo tramite l'aggiornamento di React Native si possono sopperire questi problemi, dato che il threading model è uno dei principali indiziati per l'aggiornamento dell'architettura.

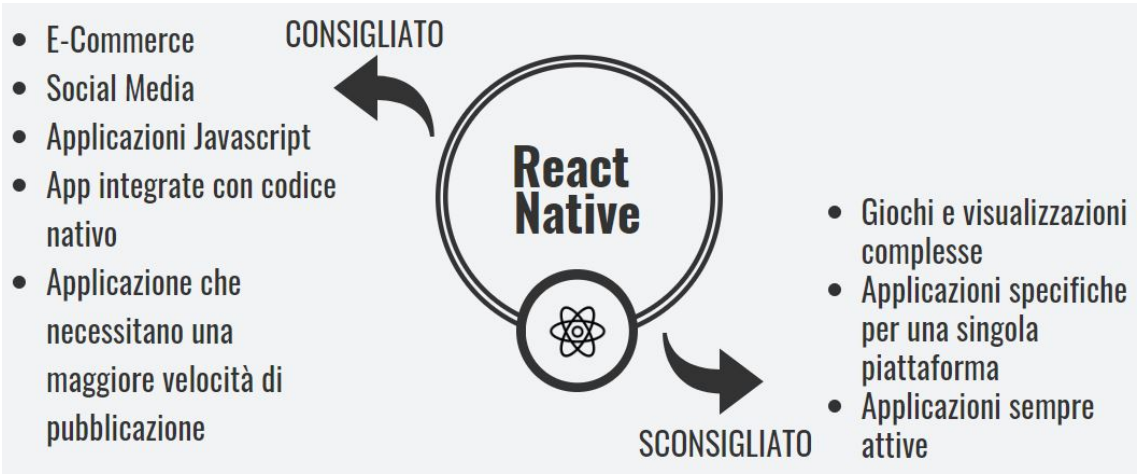

Figura 1.3: Dove è consigliato usare RN e dove non lo è

1.4 Motivi della scelta

Iniziare ad utilizzare React Native è molto semplice dato che utilizza il linguaggio di programmazione più diffuso (Javascript) e utilizza inoltre anche il gestore di pacchetti più grande al mondo (NPM), utile per includere nel proprio progetto librerie altrui. Per gli utenti che si stanno inserendo in questa tecnologia per la prima volta, questo è un sinonimo di stabilità e affidabilità poiché React Native non verrà interrotta, ma avendo una grande comunità, codici, pacchetti e nuove funzionalità saranno in costante aggiornamento. È comunque un linguaggio con una curva di apprendimento molto alta per chi proviene dal mondo web dato che javascript è un linguaggio facilmente assimilabile rispetto ai vari Objective-C, Java, ecc.

Inoltre come detto nei capitoli precedenti, la possibilità di scrivere un codice multipiattaforma causerà un risparmio di energie e risorse. Un team che deve sviluppare un'applicazione in React Native per più piattaforme si può concentrare sul miglioramento e evoluzione del codice sullo stesso progetto, mentre se si utilizza il linguaggio nativo come Java o Objective-C sarà necessario creare due progetti differenti e quindi una divisione del team iniziale e della forza lavoro a seconda delle esigenze.

Figura 1.4: Esempio di suddivisione lavoro di un team

Detto ciò il motivo principale per cui gli utenti si stanno dirigendo verso questo tipo di soluzione è principalmente il fatto che React Native è un linguaggio multipiattaforma che mantiene e sfrutta tutte quelle proprietà appartenenti ai linguaggi nativi. Le prestazioni sono minori in quanto si deve effettuare una "traduzione" del codice a nativo ma comunque React Native garantisce un'identica user experience e le prestazioni sono di poco inferiori. Questo è il motivo per cui la tecnologia

React Native viene detta pseudo-nativa, avendo una interfaccia utente (UI) sostanzialmente nativa e una business logic che viene interpretata di volta in volta. Per business logic si intende la logica applicativa che rende operativa un'applicazione, cioè il nucleo di elaborazione.

Capitolo 2

Differenze con altri linguaggi di programmazione

Per lo sviluppo di applicazioni mobile, sono presenti centinaia di soluzioni differenti che si differenziano fra loro secondo diversi fattori come portabilità, costi, popolarità, prestazioni, documentazione, ecc. Dalle diverse soluzioni presenti, se ne possono ricavare tre diverse categorie, linguaggi di programmazione nativi (Java, Objective-C,...), linguaggi di programmazione multipiattaforma ma basati su tecnologgie differenti rispetto a React Native (ibridi/web) e infine linguaggi multipiattaforma con un approccio pseduo-nativo (Flutter). Nelle sezioni successive sono elencate le varie differenze fra React Native e queste categorie di linguaggi di programmazione.

2.1 Differenze rispetto ai linguaggi ibridi/web

Le applicazioni ibride sono delle applicazioni sviluppate per soddisfare alcuni requisiti come la velocità di sviluppo e la possibilità di essere funzionante e performante su diversi dispositivi e sistemi operativi (multipiattaforma). Le app ibride sono sostanzialmente dei siti internet incorporati in una applicazioni mobile che sfrutta una semplice webview per mostrare il contenuto del sito. Il codice responsabile è formato da una collaborazione tra HTML5, CSS e JAVASCRIPT e non usa API specifiche della piattaforma ma può utilizzare alcune funzionalità native grazie ad alcuni tool, plugin come Cordova che permettono l'uso di feature come GPS e camera. Uno dei principali framework che hanno l'approccio descritto è Ionic e come React Native sono rappresentati entrambi dalla frase "Write once run everywhere", ma si differenziano fra loro per diversi motivi.

Figura 2.1: "Write once run everywhere

User Interface e aspetto

React Native si appoggia ad elementi nativi, facendo risultare la propria app un'applicazione con un "look and feel" pressochè identico a un'applicazione nativa anche tramite l'uso di svariate API messe a disposizione dal sistema, per i framework ibridi/web invece l'applicazione è comunque un sito web con una user interface molto rallentata e un peggioramento delle prestazioni dovute all'uso di un singolo "core" per l'applicazione. Un app Ionic ha a tutti gli effetti un aspetto e caratteristiche proprie di un qualsiasi sito mobile. Un'applicazione ibrida/web senza connessione dati non può funzionare al contrario di un'applicazione sviluppata in React Native.

Performance e stabilita

Le prestazioni per la maggior parte delle applicazioni, ad eccezione di app con grosso carico computazionale sono dalla parte di React Native che si avvicina a performance di applicazioni interamente native. Si può avvicinare ulteriormente dato che ha anche la possibilità di essere integrato con codice puramente nativo. Ionic dato che è una rappresentazione di una pagina web, avrà prestazioni nettamente inferiori dovute anche all'uso di tools e non direttamente di API del sistema. Per quanto riguarda la stabilità, React Native ha un solo livello tra codice e dispositivo, il livello di traduzione, mentre Ionic si dovrà appoggiare prima alla webview e successivamente a Cordova per utilizzare i plugin di comunicazione con i moduli nativi, aumentando sia i livelli che le probabilità d'errore.

Costi

Entrambe le tecnologie permettono di concentrare tutte le forze su un unico progetto/versione, ma la differenza sostanziale sta nel fatto che per quanto riguardo Ionic, un sito web è al 100% trasferibile all'interno di una nuova app (cambierà soltanto la webview), così da ridurre al minimo il tempo impiegato. Per quanto riguarda React Native invece, essendo basato su Javascript, alcune parti del codice saranno riusabili, ma comunque avrà bisogno di alcuni cambiamenti per le parti specifiche per un sistema operativo. Per i costi monetari invece, React Native è un framework completamente open-source e quindi gratuito. Per Ionic invece il prezzo dipende dai tools necessari per lo sviluppo della propria applicazione, partendo da un pacchetto gratuito per poi aumentare a seconda dei tools disponibili, maggiori funzionalità, maggior costo.

Community, supporto e documentazione

Ad oggi le applicazioni presenti sullo store, sviluppate tramite Ionic, sono in maggioranza rispetto alle applicazioni sviluppate in React Native. Questo perchè Ionic è molto facile da usare e alla portata di tutti grazie a una documentazione semplice e leggibile e un supporto base per gli sviluppatori. Le app sviluppate in Ionic, sono però applicazioni di un

altro calibro, applicazioni che solitamente hanno come destinazione un ristretto gruppo di utenza, a differenza di React Native che ha nella sua categoria applicazioni come Facebook, Instagram, Airbnb, Skype, ecc. I download di applicazioni React Native sorpassano di gran lunga i download di applicazioni in Ionic nonostante siano in minoranza.

Tabella riassuntiva differenze con Ionic

Figura 2.2: Comparazione tra React Native e Ionic

2.2 Differenze rispetto ai linguaggi pseudo-nativi (Flutter)

Negli ultimi anni nel campo del multipiattaforma, ci si sta dirigendo verso la strada del linguaggio così detto pseudo-nativo. Attraverso questo approccio si ottiene un aumento delle prestazioni dell'applicazione, dato che il rendering dei componenti è incaricato agli elementi nativi del dispositivo tramite una traduzione del linguaggio di partenza, nel nostro caso con React Native, Javascript. Ma lo sviluppo pseudo-nativo non è proprio solamente di React Native ma esistono altri linguaggi per la costruzione di applicazioni cross-platform che utilizzano quest'approccio. I primi framework nati sono NativeScript e Xamarin, il primo si basa anch'esso su Javascript, mentre il secondo pone le sue fondamenta su C#e .NET, ed essi per varie problematiche come costi, documentazione e comunità stanno finendo lentamente in disuso, favorendo React Native e Flutter.

L'accerrimo rivale di React Native nel suo stesso campo ad oggi è Flutter: un framework rilasciato due anni dopo la nascita di React Native, esattamente nel 2017, che ha come principale investitore Google. Flutter è un framework molto simile a RN su alcuni concetti, come l'approccio pseudo-nativo, ma che si fonda su basi completamente differenti. Esso si basa su due macro strati composti rispettivamente da un linguaggio di programmazione chiamato Dart e da C/C++, già da questa caratteristica si può pensare come abbia un approccio "più nativo" rispetto a React Native, che al contrario si appoggia su Javascript, linguaggio caratteristico dei vari framework web. Per differenziare in modo esaustivo le differenze tra Flutter e React Native, ho analizzato i differenti pro e contro in base a differenti parametri, i quali possono essere raggruppati in sette categorie:

Supporto e Community

Entrambi sono linguaggi di programmazione open source e essendo supportati una da Facebook e l'altro da Google avranno comunque al loro seguito un'ampia community di persone in continua crescita per entrambe le tipologie. Ad oggi avendo React Native due anni in più di vita, è

presente un gap dovuto a questa tempistica, dato che le percentuali di crescita sono pressochè uguali. Per quanto riguarda la documentazione fornita online dalle due aziende, sia RN che Flutter hanno disponibili ottime guide, tutorial e forum di supporto, ma in questo campo Flutter, per la velocità con cui vengono rilasciati pacchetti e aggiornamenti risulta la migliore. React Native dato che è un linguaggio più consolidato ha un impiego maggiore in ambito lavorativo (Facebook, Instragram, Airbnb, ecc.) rispetto a Flutter che ha troppe poche librerie e soluzioni pronte all'uso, molte cose che devono essere implementate su Flutter già esistono per React Native. Un esempio di grandi aziende che hanno scelto Flutter sono Alibaba e Google Ads.

Struttura del codice

Per quanto riguarda le prestazioni dei due framework, sono sostanzialmente molto differenti. React Native utilizza Javascript, linguaggio propriamente usato per il web, in un thread separato e grazie all'uso di un bridge comunica con i componenti nativi interessati. Flutter invece, utilizzando C/C++, si posiziona più vicino al linguaggio macchina offrendo prestazioni migliori, ad esempio la compilazione non si basa solamente ai componenti dell'interfaccia utente, ma all'intera applicazione.

A supporto della programmazione, molti tool di sviluppo per React Native sono già stati sviluppati e testati, come tool per la gestione del ciclo di vita dei componenti o gestione dello stato. In Flutter molte di queste feature non sono ancora state implementate. Una feature presente in entrambi i framework che permette di velocizzare la fase di sviluppo del codice è l'hot reloading che consiste nel poter effettuare delle modifiche all'app senza doverla ricompilare interamente.

Elementi fondamentali e UI

Flutter presenta una moltitudine di widget[1] in grado di garantire un ottimo look feel dell'interfaccia utente. Nonostante ciò Flutter non

[1]Sono l'equivalente dei componenti per React Native

è molto utilizzabile quando l'applicazione ha la necessità di avere un design con caratteristiche specifiche con riferimento alla piattaforma usata, oppure quando l'app richiede interazioni multiple con il sistema operativo utilizzando librerie native poco conosciute. React Native in questo ambito risulta essere vincente dato che è in grado di replicare un'esperienza nativa. React Native è la soluzione migliore anche nei casi in cui è previsto un uso significativo di moduli hardware del dispositivo da parte dell'app, come utilizzare la fotocamera.

Costi

Sia React Native che Flutter sono due framework open-source e perciò non esistono costi aggiuntivi per lo sviluppo. Avendo entrambi una grande community hanno un grosso contributo da parte di terzi sviluppatori per quanto riguarda aggiornamento e inserimento di nuove feature utili per lo sviluppo.

Prestazioni

Per quanto riguarda le performance delle due contendenti, si riconduce tutto al fatto che, nonostante entrambi fungano da wrapper intorno all'app vera e propria, Flutter è un linguaggio che si avvicina maggiormente al nativo senza l'utilizzo di un bridge per la traduzione del codice e librerie di terze parti come React Native. Flutter funziona in modo uniforme e comunica direttamente con le componenti native del dispositivo avvicinandosi il più possibile al linguaggio macchina con C/C++ e quindi offrire migliori prestazioni native.

Dopo tutte queste considerazioni, ad oggi il maggiormente impiegato è e rimane React Native, dato che in questo momento è il linguaggio più stabile ed avanzato fra i due, poichè Flutter ha avuto una fermata nella creazione delle librerie. In un futuro i due linguaggi potrebbero anche pareggiarsi come importanza, ma ad oggi il più rilevante e utilizzato rimane React Native.

Tabella riassuntiva differenze con Flutter

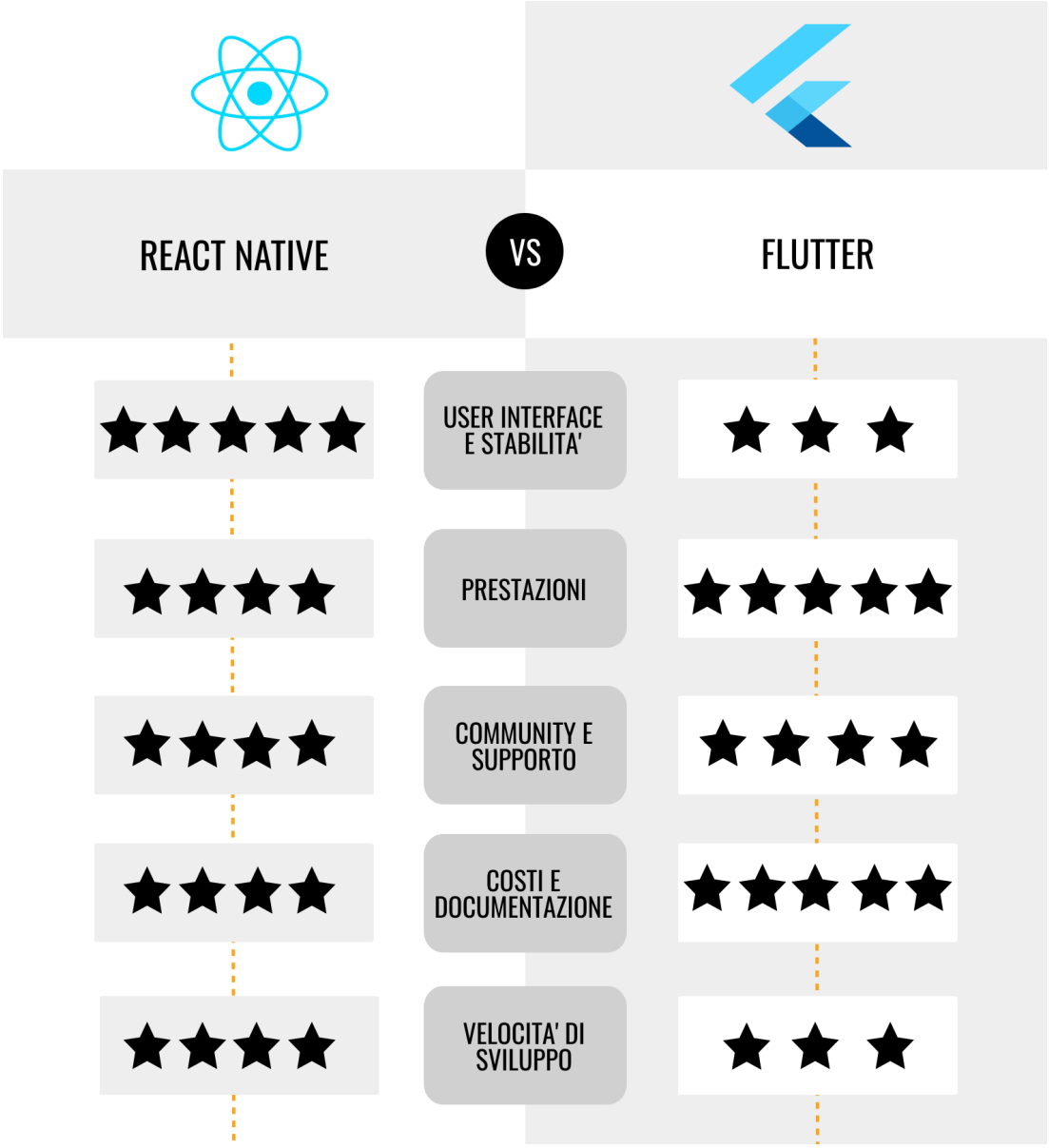

Figura 2.3: Tabella riassuntiva differenze con Flutter

2.3 Differenze rispetto ai linguaggi nativi (Android, iOS)

Nonostante un crescere dei linguaggi multipiattaforma, molti sviluppatori continuano a utilizzare un approccio completamente nativo per la creazione di alcune applicazioni. I linguaggi nativi sono molteplici e sono caratteristici di un solo sistema operativo, i più famosi sono Java per Android e Objective-C per iOS.

Il linguaggio nativo è ancora ampiamente utilizzato grazie alla sua robustezza del linguaggio, prendendo come esempio Java, è un linguaggio fortemente tipato e compilato al contrario di Javascript che invece è un linguaggio debolmente tipato e interpretato. Questo significa che è necessaria una maggiore attenzione per quanto riguarda la scrittura di applicazioni con Javascript dato che il compilatore non aiuta lo sviluppatore nello scovare gli errori. La differenza fra i due linguaggi è compensata dal fatto che Javascript è il linguaggio di programmazione più conosciuto e usato al mondo e quindi risulta semplice trovare buoni sviluppatori in grado di utilizzarlo, il che sopperirà alla mancanza del linguaggio rispetto ai linguaggi nativi.

Altre ragioni del perchè si continua ad utilizzare linguaggi nativi sono il supporto da parte della comunità e un completo accesso a tutte le API o comunque feature disponibili del sistema operativo. Per quanto riguarda la prima ragione, essendo innanzitutto linguaggi nati molti anni prima e avendo un largo impiego sul mercato, i linguaggi nativi hanno ottenuto un seguito maggiore rispetto a React Native dopo solo 5 anni dalla sua creazione. Ovviamente è un dato destinato a ribaltarsi ma ad ora le persone che ne fanno parte e creano librerie di supporto rimangono in maggioranza. L'utilizzo di API per un linguaggio mobile è fondamentale e tramite un approccio nativo sono tutte a disposizione dello sviluppatore e quindi anche tutte le funzionalità che il sistema operativo offre, React Native in questo campo è ancora un passo indietro dato che non ha la libertà di usare tutte le API disponibili o comunque elementi specifici o differenti applicazioni del sistema operativo in questione.

Le performance e la user experience sono due caratteristiche importanti per la valutazione di un'applicazione e sono due dei vantaggi dell'utilizzo di un linguaggio nativo. E' possibile usare thread asincroni differenti per lo svolgimento di calcoli complessi o feature avanzate, cosa che invece non è possibile con Javascript di React Native dato che lavora su un singolo thread dedicato e non può performare differenti tasks asincroni, utili per lo sviluppo di applicazioni complesse. Nonostante l'uso del framework RN garantisca una ottima user experience, essa non sarà mai come la UI creata da un linguaggio nativo dato che creare interfacce utente complesse con visualizzazioni personalizzate, modelli di navigazione, animazioni, ecc. risulta difficile con React Native. I linguaggi nativi essendo sviluppati per una piattaforma specifica sono invece più adatti allo sviluppo di queste dalla UI per il sistema operativo.

Malgrado tutti questi fattori siano a favore dei linguaggi nativi, ci sono svariati motivi per cui React Native possa risultare una loro valida alternativa, a partire dalla possibilità di scrivere un codice unico per tutti i sistemi operativi. Non potendo soddisfare questa proprietà utilizzare un approccio nativo risulta maggiormente dispendioso in termini di tempo, è necessario sviluppare due applicazioni (una per Android e una per iOS), e in termini di costo, è necessario avere un altro team di sviluppo che si occupi dell'altra versione. Un altro importante fattore riguarda la manutenzione del codice, sviluppare un solo codice permette di controllare e correggerne solo uno, sviluppare due o più versioni necessità un controllo di entrambe. Se l'intenzione è quella di sviluppare un'applicazione semplice e la necessità è averla il prima possibile, React Native scavalca così i linguaggi nativi nelle gerarchie.

La scelta fra i due differenti approcci da usare in un progetto è rimandata alle effettive necessità di cui l'applicazione ha bisogno: se l'intenzione è quella di ottenere un app complessa, incentrata su una perfetta user experience, che utilizza funzionalità specifiche di un sistema operativo o che comunque è propria di quel sistema, allora la decisione va in direzione di un approccio nativo, se invece l'intenzione è quella di sviluppare un app semplice, direzionata su più sistemi operativi differenti, con un budget iniziale limitato e necessità di pubblicare e correggere il codice il più velocemente possibile, l'utilizzo di React

Native risulta più adatto. Dato che il mercato è pieno di applicazioni semplici e sviluppate per svolgere un'unica funzionalità, React Native si sta prendendo una grossa parte del mercato.

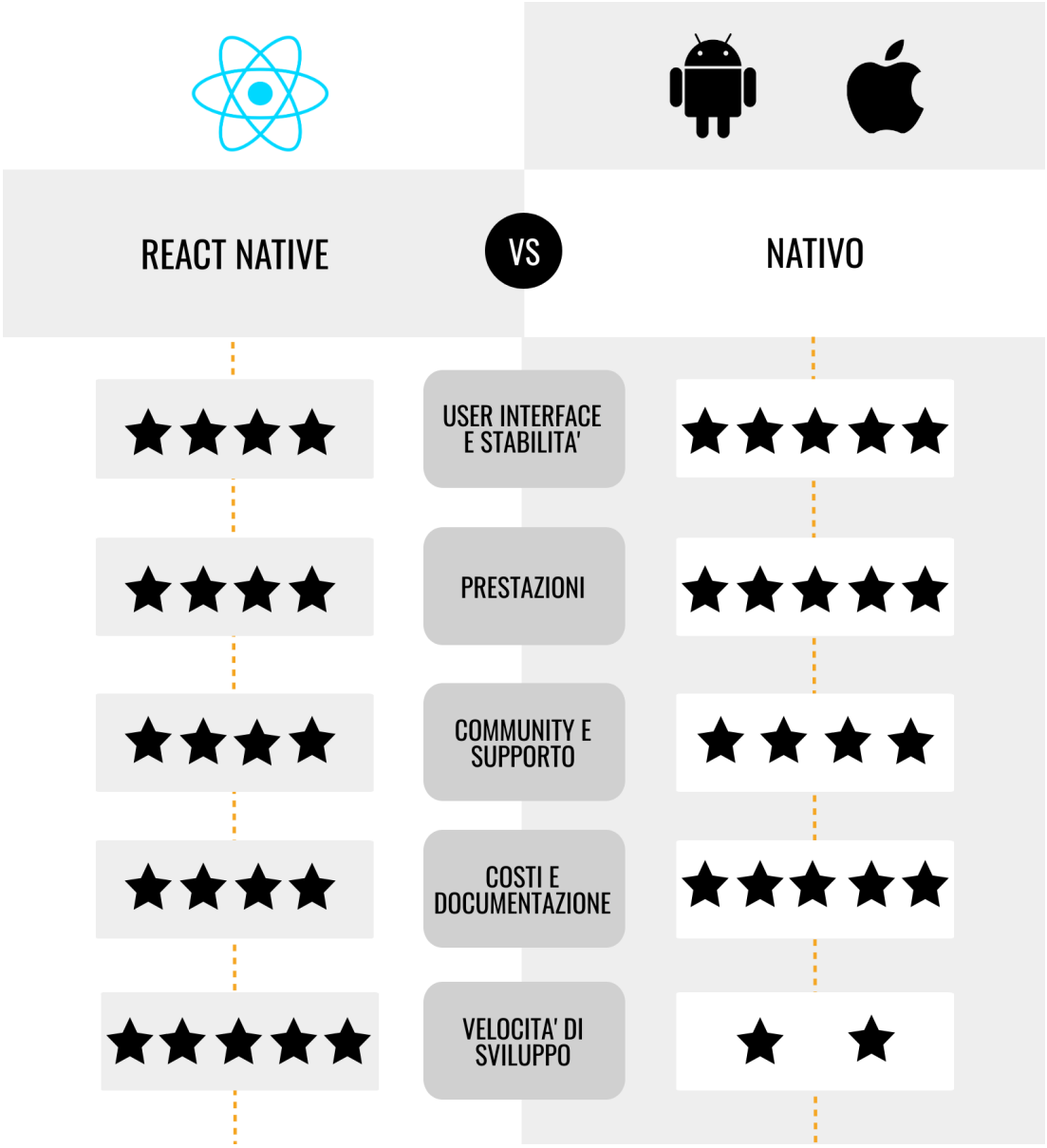

Figura 2.4: Tabella riassuntiva differenze con linguaggi nativi

2.4 Vantaggi e svantaggi

2.4.1 Vantaggi

Dopo aver analizzato il funzionamento di React Native e le differenze rispetto ad altri tipi di linguaggio di programmazione, è necessario elencarne gli effettivi vantaggi che può portare l'uso di questa tecnologia.

Come detto nei capitoli precedenti, l'ideatore di React Native è una multinazionale come Facebook, il che vuol dire la presenza di un'ampia community in continuo aumento e framework costantemente aggiornato e rinnovato. Nonostante ciò React Native è un framework completamente gratuito, dato che è open-source e quindi ogni singolo utente può modificare la propria parte di codice a seconda delle proprie esigenze, e renderla pubblica online aiutando altri eventuali sviluppatori che ne possono trarre vantaggio. Date queste ragioni molti sviluppatori iniziano a programmare in React Native, ma il motivo principale spesso è la sua semplicità, derivante da Javascript (linguaggio conosciuto da qualsiasi programmatore web). React Native ha una curva di apprendimento molto alta il che diminuisce i periodi di formazione.

Oltre alla semplicità di sviluppo, React Native è stato creato per essere duttile su più fronti, specialmente nel campo del multipiattaforma. Essendo un framework cross-platform, con una riusabilità del codice da piattaforma a piattaforma del 85/90%, anche le tempistiche e le velocità di sviluppo in un progetto risultano minori, team di sviluppo si possono dedicare ad un unico progetto e non dividersi il carico di lavoro per versioni differenti. Risparmio di lavoro e tempistiche porta a una riduzione dei costi.

I precedenti sono vantaggi generali, ma sono presenti anche alcuni vantaggi in termini di prestazioni per quanto riguarda un'applicazione sviluppata in React Native. Innanzitutto la user interface è pressochè identica alla UI di un'applicazione completamente nativa il che è essenziale per una app che deve essere introdotta in un mercato con altrettante applicazioni che svolgono lo stesso compito. Basandosi su Javascript è molto veloce e semplice da sviluppare e inoltre presenta una moltitudine di componenti e feature già sviluppate da terzi e messe

a disposizione dalla community, il che velocizza il tempo di sviluppo. La feature più importante è l'hot realoading, che permette di visualizzare modifiche dell'app, specialmente in fase di test, senza dover ricompilare l'intero progetto. Le performance, avendo un approccio pseudo-nativo, non sono ottime come un'applicazione completamente nativa anche se ci si avvicinano molto tramite l'utilizzo delle API. Per ottenere prestazioni ancora più vicine a prestazioni di applicazioni native, tramite React Native è possibile includere del codice nativo per determinate funzioni e quindi utilizzare funzioni specifiche. Detto ciò il ciclo di sviluppo risulta molto rapido e il risultato è un'applicazione nativa tradotta tramite una interpretazione di una business logic scritta in Javacript.

Una ulteriore motivazione che può portare a scegliere React Native in fase di sviluppo di una app è il suo effettivo uso in applicazioni che fanno già parte del mercato. Aziende come Facebook, Instagram, Airbnb, Skype, Tesla e molte altre hanno scelto di investire in questo framework per la loro applicazione, ponendo grossa fiducia in esso e contribuendo alla sua crescita.

Figura 2.5: Loghi di aziende che utilizzano React Native per la loro applicazione.

2.4.2 Svantaggi

Nonostante i vantaggi del scegliere React Native siano molteplici, sono presenti alcune ragioni per il quale React Native non è l'ideale. Supportato da Facebook ma non altrettando da Apple e Google, quest'ultimo ideatore di Flutter, i quali continuano ad aggiornare i propri sistemi operativi Android e iOS inserendo nuove funzionalità e feature, senza però fornire supporto al framework il quale deve sopperire a questo gap tramite aggiornamenti da parte di terzi con tempistiche chiaramente più lunghe. Per questi motivi React Native è in continua crescita, e una continua crescita significa che app già sviluppate necessitano di un mantenimento e aggiornamento del codice proporzionale alla crescita di React Native. Questo può creare preoccupazioni riguardo la longevità del progetto, ma dato che altrettanti colossi come Facebook, Instagram, ecc. lo supportano deve tranquillizzare gli sviluppatori che scelgono di iniziare a creare app in React Native.

Basandosi su React e Javascript, oltre a prenderne tutti i pregi, ne eredita anche i difetti, perciò da React avremo approccio principalmente orientato al web, a differenza di un approccio per le applicazioni native, il che non garantisce linee guida di sviluppo dei dispositivi e crea un senso di confusione per l'utente utilizzatore. Da Javascript invece, il difetto principale che React Native eredita è l'utilizzo di un linguaggio debolmente tipato e perciò ne consegue che è più facile il verificarsi di errori ed è più difficile la loro correzione. A differenza di ciò i linguaggi completamente nativi sono per la maggior parte fortemente tipati per ridurre al minimo la possibilità di errore.

Essendo React Native un linguaggio pseudo-nativo avremo un peggioramento delle prestazioni, nonostante la UI sia nativa, ed è dovuto ad una interpretazione della business logic. Il bridge di comunicazione tra javascript e le componenti native, il poco supporto per le API, l'impossibilità di inserire alcune componenti specifiche provocano un peggioramento delle prestazioni che rimane comunque relativo a seconda dello scopo e del carico di lavoro per la quale è stata ideata l'applicazione. Avendo questi difetti React Native è sconsigliato per alcuni tipi di applicazioni, giochi e multimediali, e perciò il suo uso ne risulta ridotto nonostante gli sviluppatori si stiano muovendo per colmare il gap.

Utilizzando React Native si ha a che fare nella maggior parte dei casi con librerie sviluppate da terzi, le quali possono non essere perfette e causare problemi all'applicazione. Inoltre l'utilizzo di queste librerie porterà a una dimensione dei file di installazione (APK per android e IPA per iOS) maggiore rispetto che alla controparte nativa.

Dopo aver analizzato i vantaggi e gli svantaggi di questo linguaggio si può capire il perché gli sviluppatori e le compagnie preferiscano React Native rispetto ad altri linguaggi. Gli sviluppatori scelgono React Native per la sua alta "leggibilità" (dato che è basato su JavaScript), facilità di apprendimento, una grossa comunità di supporto e inoltre la possibilità di non ricompilare il codice ad ogni modifica effettuata. Le compagnie scelgono React Native invece, per differenti motivi ma altrettanto validi: come ad esempio la riduzione del tempo di lavoro su un'applicazione, dato che è sufficiente crearne una soltanto grazie al cross-platform, e di conseguenza anche la riduzione dei costi, dovuta al minor tempo impiegato e team di sviluppo composti da un numero minore di persone.

Nonostante ciò, React-Native non è perfetto e può essere migliorato, cercando di migliorare le prestazioni e aumentando affidabilità e qualità dei componenti di terze parti.

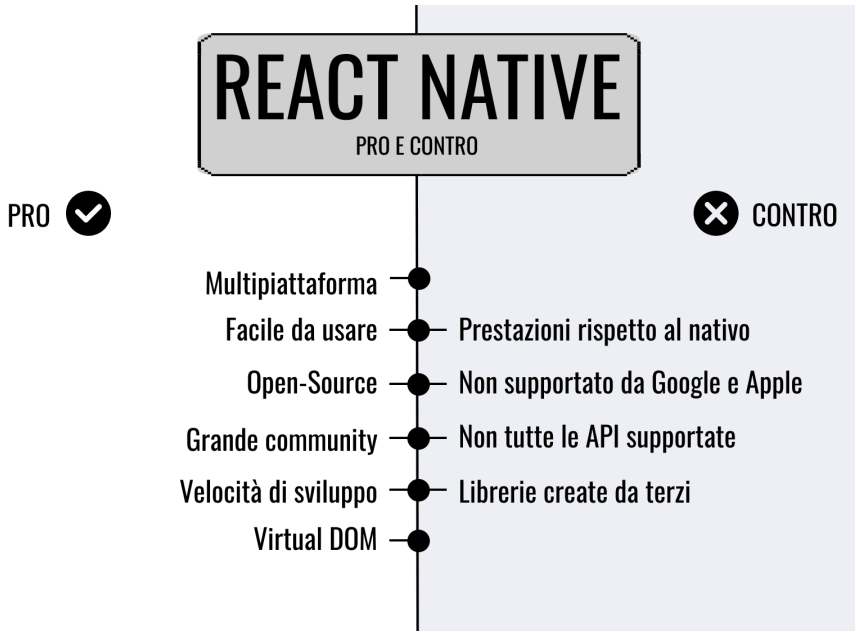

Figura 2.6: Pro e contro di React Native

2.5 Futuro di React Native

React Native, come visto nei capitoli precedenti, è in continua crescita nella community di sviluppatori, ed è sufficiente dare un'occhiata ai Google Trends[2] durante gli anni per capire che l'interesse verso questa tecnologia è sempre in costante sviluppo.

Facebook negli ultimi anni sta cercando il più possibile di coinvolgere la propria community, creando repository apposite per tenersi aggiornati sulle ultime release (react-native-relaease) oppure repository utili per scambio di idee e proposte di aggiornamento da parte della community. Prossimamente i principali sviluppatori di React Native sono intenzionati a creare una serie di standard per i vari package e repos presenti su GitHub di modo da aiutare e accrescere ulteriormente la community, pilastro portante del linguaggio React Native.

In termini di architettura gli ideatori di React Native stanno cercando di apportare delle modifiche per rendere il framework più flessibile e adattabile con l'infrastruttura nativa delle applicazioni ibride. I principali cambiamenti saranno in diverse sezioni come il threading model (semplificando il Virtual DOM con la creazione di meno thread di supporto), rendering asincrono (per permettere la creazione di richieste di rendering multiple e semplificare la gestione delle richieste asincrone) e creazione di un bridge più veloce e immediato (il bridge gestisce le chiamate dirette tra il nativo e Javascript). Grazie a queste modifiche le prestazioni di React Native aumenteranno ancora anche se ad oggi non è ancora possibile lavorare esattamente a livello nativo ma con il passare degli anni e il supporto della community la differenza fra essi si assottiglierà.

In conclusione, le modifiche all'architettura, l'ampliamento e il coinvolgimento di tutta la community, l'utilizzo da parte delle maggiori imprese nel settore, ci portano a dire che React Native è il futuro ed è un linguaggio in continua crescita supportato sempre di più da diverse aziende.

[2]Google Trends è uno strumento, basato su Google, che permette di conoscere la frequenza di ricerca sui motori di ricerca del web di una determinata parola o frase

Parte progettuale con Moltosenso

Sviluppo di due applicazioni, Tag Reader App e Cidiu4u App

Capitolo 3

Tag Reader APP

3.1 Spiegazione del progetto

Il primo progetto al quale ho preso parte, consiste nello sviluppo di una applicazione completamente in React Native in collaborazione con Moltosenso s.r.l. e Cidiu Servizi Spa, società che ha per oggetto i servizi di igiene urbana costituiti dall'insieme delle operazioni di igiene urbana, raccolta di rifiuti solidi urbani, raccolta differenziata, trasporto, conferimento e tutte le attività connesse, accessorie e complementari . Essa si occupa dei comuni limitrofi a Torino, per un territorio totale di 341 kmq.

In questo caso specifico il progetto dell'applicazione Tag Reader App si sviluppa per risolvere un problema presente in fase di censimento dei siti di raccolta differenziata dei rifiuti presenti sul territorio. Ogni contenitore posizionato nel territorio è dotato di un tag radio (RDID) contenente un ID caratteristico e unico, ed è utilizzato in fase di censimento e conferimento. Nella prima fase il tag è usato per ottenere la localizzazione, tipologia e volume del bidone. I dati, una volta raccolti vengono inseriti in un database online il quale sarà utile per tenere traccia dei vari spostamenti che si possono creare con il passare degli anni o creare statistiche e/o grafici sulla loro posizione e utilizzo nel territorio. Il tag è utilizzato anche in fase di svuotamento del contenitore, per attribuire lo scarico di un determinato volume di rifiuto a un determinato ID del bidone, e tenerne traccia su un database online. Si ottiene così un controllo sull'andamento dei vari conferimenti annuali

in una particolare zona.

Tag Reader App è un'applicazione che si occupa di risolvere la prima fase, ovvero la fase di censimento. Ad oggi la lettura dei tag viene effettuata tramite un dispositivo hardware creato per l'occorrenza, il quale presenta un lettore di tag posto all'estremità per leggere l'ID di ogni specifico contenitore, modulo di geolocalizzazione, uno schermo LCD per la visualizzazione dei dati (valore del TAG, latitudine e longitudine, data e ora) e un comunicatore bluetooth seriale. Quest'ultimo è incaricato di comunicare i dati ottenuti al computer di supporto che si occuperà, una volta controllati e inseriti di nuovi, di inviarli al database online. Sono presenti inoltre alcuni pulsanti fisici per il reset e lettura indipendente del solo valore GPS ed una batteria per rendere il dispositivo portatile. Tutte queste componenti sono inserite e collegate in un case di plastica di dimensioni all'incirca 10 cm x 20 cm x 5 cm, che diventa a tutti gli effetti un dispositivo hardware.

L'applicazione Tag Reader App è stata pensata per apportare delle migliorie alla fase del censimento e inoltre sfruttare i vari vantaggi che può portare l'utilizzo di un linguaggio come React Native, elencati nel capitolo precedente, in un contesto lavorativo e di utilizzo quotidiano.

La creazione di Tag Reader App è stata effettuata per soddisfare un bisogno concreto di un'azienda, un'esigenza, ma essendo il censimento una fase caratteristica di qualsiasi azienda in qualsiasi campo, l'applicazione è stata sviluppata con un impronta general purpose di modo che possa essere velocemente riadattata per qualsiasi richiesta non inerente alla spazzatura e i bidoni. Essa diventa perciò un'applicazione utile e necessaria per il censimento dei bidoni della spazzatura ma riutilizzabile in un qualsiasi campo con poche e veloci modifiche alla sola activity di visualizzazione e modifica dei dati.

3.2 Quale bisogno soddisfa

In termini di funzionalità in linea teorica il dispositivo non presenta problemi, legge i dati dal contenitore e li invia ad un computer di supporto che li elabora e successivamente li invia al database, ma in un contesto pratico e operativo si sono presentati alcuni problemi. Il disguido principale che è stato riscontrato riguarda le dimensioni sia del pc di supporto che del dispositivo hardware che risultano essere ingombranti per il compito che l'operatore deve svolgere.

I bidoni sono alla portata di chiunque e i tag, posizionati all'interno lungo la parete o comunque in un luogo accessibile inizialmente, possono spostarsi in luoghi scomodi o avere degli intralci che rendono difficile la lettura del tag all'operatore di nettezza urbana, specialmente se il dispositivo necessità l'utilizzo di entrambe le mani date le sue misure. Come risoluzione a questo problema si è pensato di ridurre le dimensioni effettive del dispositivo, eliminando lo schermo LED (occupava la maggior parte dello spazio), per destinare la parte di visualizzazione dati all'applicazione Tag Reader App e perciò ridurre anche le dimensioni della batteria la quale veniva consumata principalmente dallo schermo. La comunicazione tra l'app in React Native e il dispositivo "ridotto" avviene comunque tramite una comunicazione in bluetooth seriale come avveniva in precedenza con il computer di supporto.

Il computer di supporto è un dispositivo a parte, relativamente tascabile, il quale raccoglie i dati tramite bluetooth, li manipola e li reinoltra modificati, se necessario, al cloud che si occupa di immagazzinarli. Essendo un computer portatile esso è posizionato sul camion di raccolta ed è gestito da un altro operatore. Con la possibilità di visualizzare e eventualmente modificare i dati direttamente dall'applicazione creata, il computer di supporto potrebbe risultare superfluo dato che è possibile eseguire tutte le sue funzionalità direttamente dall'app. Questo è un vantaggio in termini di tempistiche dato che ogni operatore potrà autogestirsi, tenendo in una mano il nuovo dispositivo di lettura TAG tascabile, e nell'altra il proprio telefono mobile con all'interno Tag Reader App, evitando così una comunicazione tra due operatori che ovviamente provoca ritardi e incomprensioni tra chi gestisce il dispositivo e chi il computer di supporto. Questa comunicazione diventerà superflua

e perciò dotando ogni operatore di un lettore di TAG e dell'applicazione Tag Reader App si potrà anche ottimizzare il lavoro.

Tutti gli operatori sono in possesso di un telefono cellulare, personale o aziendale, e quindi l'utilizzo dell'applicazione Tag Reader App non comporterebbe l'acquisto di un ulteriore dispositivo per la gestione dei dati ma anzi porterebbe a un risparmio notevole di tutti i computer di supporto che sono essenziali per il censimento ad oggi e che quindi possono essere riutilizzati per altri scopi senza un ulteriore spesa.

Fino a qui la scelta di creare una applicazione non riguarda React Native perchè sia applicazioni Android che applicazioni sviluppate in iOS sono in grado di svolgere i compiti precedentemente descritti, ma non potendo basarsi su una versione mobile standard per ogni operatore, non si può essere a conoscenza dei vari modelli in possesso degli operatori di nettezza urbana, si è voluto iniziare a sviluppare con un linguaggio principalmente multipiattaforma, ma comunque innovativo e con un futuro davanti. Analizzati i punti a favore dei vari linguaggi innovativi, si è scelto React Native grazie al suo essere un linguaggio cross-platform il che garantisce una non preoccupazione per gli smartphone posseduti dai dipendenti e un minore tempo di sviluppo, dato che lo sviluppo per la piattaforma Android e iOS è lo stesso.

3.3 Sviluppo dell'app

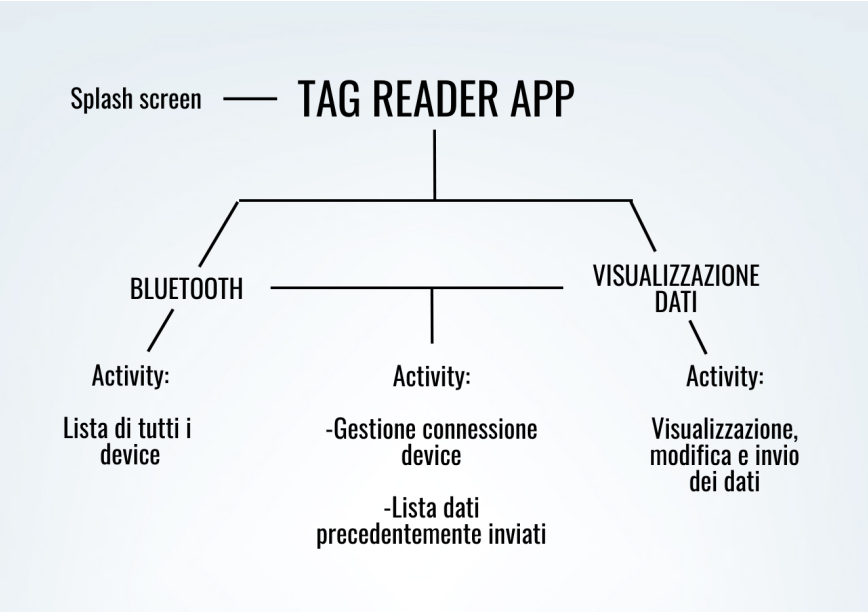

Figura 3.1: Mappa concettuale funzionamenteo Tag Reader App

L'applicazione Tag Reader App, come detto in precedenza, comunica con il dispositivo hardware tramite una comunicazione bluetooth seriale e si incarica di visualizzare i dati ricevuti, modificarli ed eventualmente inviarli al database online. Le funzionalità dell'app si possono quindi dividere in queste due macrocategorie (bluetooth e dati) ed ogni activity può appartenere a ciascuna di essa oppure ad entrambe. Il funzionamento di Tag Reader App è lineare, si passa da un'activity all'altra senza la possibilità di prendere un ramo differente come spesso accade nella maggior parte delle applicazioni sul mercato, e questo è un fatto voluto dato che la sua principale ed unica funzione è l'inserimento di dati relativi al bidone in questione in un database online per il censimento.

Per sfruttare al meglio le potenzialità di React Native si sfruttano alcune librerie create direttamente da Facebook e altre librerie create da terze parti rese disponibili su Github o altri siti di supporto. Per la loro installazione si utilizza un gestore di pacchetti NPM che va ad

installare nel progetto pacchetti pubblici o privati tramite un database online.

L'applicazione si sviluppa in quattro differenti activity che comunicano fra di loro tramite la libreria *react-navigation*. Questa libreria consente all'app di passare da un'activity all'altra e gestire la cronologia di navigazione, si basa sul concetto di stack navigator nel quale ogni nuova activity entra sempre in cima allo stack e passa in background quando una nuova activity viene creata. La rimozione di un'activity equivale all'eliminazione dallo stack di navigazione mentre il passaggio in background equivale a scendere di livello nella pila dello stack.

Come detto in precedenza l'applicazione si sviluppa in quattro differenti activity: Splash Screen, Lista Devices, Device Page, Device Reader.

3.3.1 Splash Screen

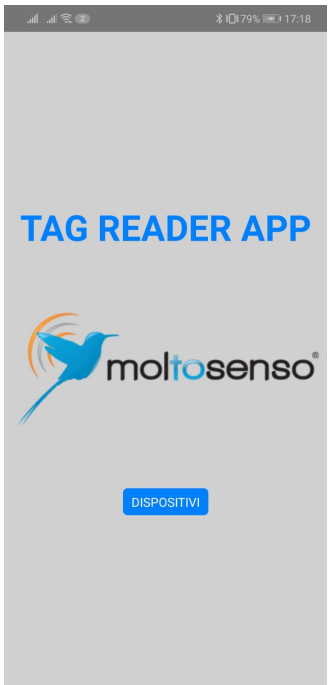

Figura 3.2: Schermata Splash Screen

Schermata iniziale dell'applicazione, visibile solamente al primo avvio dopo l'installazione dove sono presenti unicamente il nome dell'app e il logo e la sola azione possibile è passare alla schermata successiva, ovvero l'activity contenente la lista dei dispositivi bluetooth associati al telefono cellulare. Questa è una pagina anche di configurazione dato che ricaverà l'url del sito responsabile per l'invio dei dati al database da un file di configurazione scritto in JSON e lo salverà come variabile globale.

3.3.2 Lista Devices

La seconda activity presente si chiama Lista Devices ed è utile per la rappresentazione dei dispositivi bluetooth associati al telefono. Si deve selezionare il dispositivo Tag_Reader_Esp[1], controllando che il MAC address del device sia corretto dato che si potrebbero utilizzare diversi lettori di tag, e una volta selezionato l'app si sposterà sull'activity personale del dispositivo. Essendo general-purpose, se l'applicazione fosse personalizzata per la modifica e invio di dati differenti, sarebbe sufficiente selezionare un dispositivo differente, connetterlo, e il suo utilizzo sarebbe identico anche se in un contesto differente.

Basandosi principalmente sul bluetooth, l'applicazione utilizza una libreria *react-native-bluetooth-serial-next* la quale gestisce tutti i vari protocolli di comunicazione bluetooth seriale: accensione e spegnimento del bluetooth, connessione e disconnessione con un dispositivo, visualizzazione dei dispositivi associati e non, e per ultimo cosa, crea vari listener in ascolto su differenti eventi come l'arrivo di un dato da parte del dispositivo connesso, o la disconnessione con un dispositivo. Con questa libreria è possibile sia leggere che scrivere sul canale di comunicazione creato con un device, ma per la funzione di Tag Reader App, si utilizza solamente la fase di lettura.

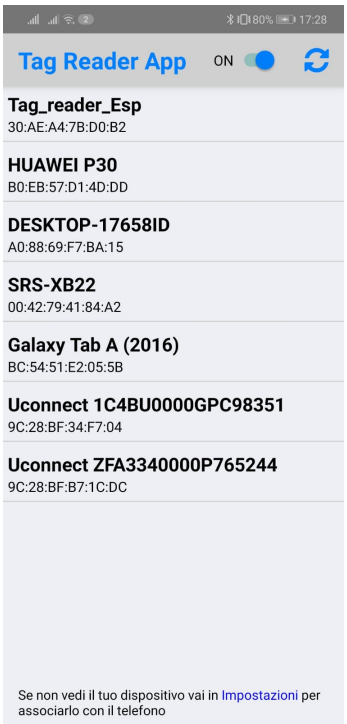

Figura 3.3: Schermata List Devices

In questa schermata sono possibili ulteriori funzionalità come la possibilità di accendere o spegnere il bluetooth, pensata principalemente per il caso in cui il bluetooth fosse spento (Tag Reader App è un'applicazione che si basa su bluetooth, non ci sarebbe ragione di spegnerlo) oppure riaggiornare la lista dei dispositivi associati nel caso in cui il telefono

[1]Il dispositivo in questione è il dispositivo hardware creato per il censimento

si fosse associato con il lettore tag solamente dopo l'avvio dell'applicazione. Nel caso in cui il device non fosse presente, è necessario settarlo dalla pagine delle impostazioni bluetooth e per poter passare immediatamente ad essa è presente un link in basso alla schermata, che grazie alla libreria *react-native-android-open-settings* , permette di ridirezionare l'utente direttamente sulla pagina di impostazioni bluetooth del proprio dispositivo. Questo è un caso che descrive una proprietà di React Native, una riusabilità del codice del'80/90%, dato che questa libreria funziona unicamente per Android. Se si volesse avere una portabilità del 100% e quindi anche per iOS sarebbe sufficiente implementare una nuova libreria o sostituire quella esistente con un'altra che implementa la funzionalità per tutti i sistemi operativi. E' possibile personalizzare il codice a seconda del sistema operativo presente tramite un semplice if e questo permette l'utilizzo di librerie specifiche per Android e librerie specifiche per iOS.

3.3.3 Device Page

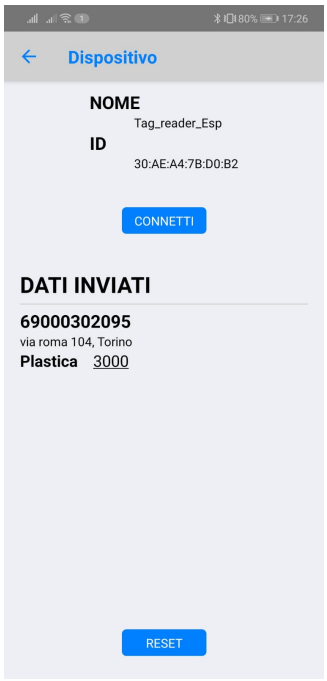

Figura 3.4: Schermata Device Page

Questa activity è l'unica che sfrutta sia le funzionalità del bluetooth che la visualizzazione dei dati. Dopo aver selezionato uno specifico device nella schermata precedente, tramite altre opzioni della libreria *react-native-bluetooth-serial-next* sapendo nome e mac address (passati come parametro dall'activity precedente), è possibile connettere l'applicazione Tag Reader App con il device incaricato di leggere i tag. Connettendosi fra loro si verrà a creare un ponte di comunicazione bluetooth sul quale si può scrivere in qualsiasi momento, e quando il lettore tag scrive le informazioni ricavate (tag, posizione, data e ora) l'applicazione attiverà un evento come risposta, dato che è in continuo ascolto sul canale creato. Una volta connesso il device è possibile anche disconnettersi per potersi collegare con un altro dispositivo oppure anche quando il lavoro di censimento è terminato.

Oltre alla funzione relativa al bluetooth, è presente una lista, inizialmente vuota, creata tramite una FlatList[2] che si aggiorna con i parametri di ritorno ogni qual volta che un dato viene inviato al database. Si ottiene così una lista dei dati precedentemente inviati mentre i dati cancellati o per qualsiasi motivo non inviati, non faranno parte di questa lista. E' possibile svuotare in qualsiasi momento la lista.

[2]E' un componente di Javascript che permette di gestire una lista di dati in entrata e renderizzarli uno ad uno sullo schermo.

3.3.4 Device Reader

Una volta attivato l'evento nell'activity precedente, i dati vengono letti e poi passati come parametro a questa activity che li elabora, manipola ed invia. I dati ricevuti vengono salvati in una stringa comune, splittati[3] e posizionati in un array, utile per visualizzare il tag a schermo e controllare che esso non sia già presente in database[4], nel caso diventa visibile un dialog che lo rende noto. Per quanto riguarda la posizione, essa viene inviata comunque al telefono cellulare e l'applicazione nel codice ha la possibilità di utilizzarla, ma per ora, dato che il GPS del telefono cellulare è di certo più preciso del GPS del dispositivo hardware, si utilizzano i dati relativi alla localizzazione tramite il GPS dello smartphone. Latitudine e longitudine saranno i valori di inizializzazione della mappa, di modo da centrarla immediatamente sulla posizione dell'operatore, si utilizza *react-native-maps*[5].

Figura 3.5: Schermata Device Reader

Tramite una libreria, *nominatim-geocoder*, è possibile, grazie a una richiesta online con una API, venire a conoscenza del nome della via partendo da latitudine e longitudine,e qual'ora non fosse la posizione corretta inserendo nel TextInput via, civico e città manualmente dall'operatore, si possono ricavare latitudine e longitudine con un'altra richiesta API con parametro la stringa inserita, grazie alla funzionalità

[3]Separare una stringa in un array di stringhe tramite un delimitatore passato come input

[4]Si utilizza una API già presente nel sistema precedente

[5]E' la libreria che gestisce le mappe, si utilizza il provider Google, usando Google Maps, tramite l'uso di una Google Api Key

inversa della libreria utilizzata,e posizionare la mappa nella posizione scelta.

Data e ora vengono visualizzati a schermo per permettere un confronto all'operatore e capire che i dati visualizzati sono effettivamente legati alla lettura in quel momento e non dati precedentemente inviati ma rimasti in sospeso.

Questa appena descritta è la parte general-purpose dato che per un qualsiasi censimento, dati relativi a tag, posizione, data e ora sono essenziali, sono presenti però altre funzionalità proprie di questo progetto. La prima è l'inserimento tramite un dialog dell'ID del sito, necessario per capire a quale raggruppamento i bidoni della spazzatura analizzati appartengono, l'inserimento è regolato da una RegEx[6] per porre dei vincoli per l'inserimento nel database. E' necessario inoltre inserire la tipologia del bidone (carta, plastica, vetro, differenziata), e a seconda del tipo, facendo una specifica richiesta http al sito dell'azienda, si ottengono i valori dei possibili volumi inerenti al tipo del bidone che si possono trovare. L'ultimo dato da inserire relativo al bidone si ottiene da uno switch per stabilire se il bidone preso in considerazione è aperto o chiuso.

Tutti questi dati vengono raccolti in variabili settate come stato, quindi mutevoli, e tramite il bottone Invia, saranno inviati al database online descritto dall'url contenuto nel file di configurazione dall'activity iniziale. Una volta inviati i dati, tag, volume, tipologia, posizione vengono reinviati all'activity precedente che li salva nella sua lista. Ritornati all'activity propria del device l'applicazione Tag Reader App si rimetterà in ascolto fino all'arrivo di un nuovo dato o a una futura disconnessione.

[6]E' una espressione regolare ovvero una sequenza di simboli che identifica un insieme di stringhe

3.4 Scelte per l'applicazione

Il funzionamento principale dell'applicazione Tag Reader App consiste nell'ottenere dati da un dispositivo bluetooth, visualizzarli, modificarli e inviarli ad un database online, in questo caso constestualizzato a un censimento di bidoni della spazzatura per una azienda. L'obbiettivo principale è quello di sviluppare un'applicazione general-purpose, riadattabile a qualsiasi contesto con poche modifiche di codice.

Le modifiche da attuare riguardano il file di configurazione dato che contiene lo specifico url per l'invio dei dati, e questo può essere fatto solamente dallo sviluppatore e non dall'utente, e l'activity Device Reader. La schermata Device Reader necessità di essere cambiata dato che presenta caratteristiche dell'oggetto che si vuole censire ed è necessario modificarle secondo le richieste del contesto per cui viene sviluppata. La parte di gestione bluetooth rimane invariata dato che funge da contorno allo scambio di dati con il lettore di tag e non manipola in alcun modo i dati ricevuti.

Le azioni possibili sono facilmente intuibili grazie all'utilizzo di bottoni, switch e icone, queste ultime ottenute grazie all'utilizzo di due librerie *react-native-elements* e *react-native-vector-icons*.

L'applicazione dovendo essere usata principalmente in ambito lavorativo, non permette di passare a una vista in landscape ma soltanto in portrait mode. Basandosi per una parte sul bluetooth e per la restante sullo scambio dati, è possibile utilizzarla senza connessione dati per quanto riguarda la prima parte ma è imbossibile utilizzarla per l'invio dei dati sul database, risultando perciò inutile nel complesso.

L'applicazione ha poche activity e poche funzionalità ma essenziali per lo scopo che si era preposto ad inizio progetto. In ogni activity le azioni da eseguire sono facilmente intuibili e semplici da effettuare dato che l'operatore che utilizzerà Tag Reader App sarà sul posto di lavoro in condizioni probabilmente non perfette per l'utilizzo di uno smartphone e di un dispositivo per la lettura dei tag (ad esempio in questo caso, i bidoni si trovano a bordo strada e nel traffico torinese può non rivelarsi una condizione ottimale)

Capitolo 4

Cidiu4u APP

4.1 Spiegazione del progetto

Per la parte implementativa, oltre al progetto descritto in precedenza, si è sviluppata un'altra applicazione completamente in React Native, partendo dalla versione dell'applicazione nativa già esistente sullo store. L'applicazione si chiama Cidiu4u ed è stata sviluppata da Moltosenso s.r.l e CIDIU SpA, essa si dedica alla gestione della raccolta rifiuti di tutti i comuni che sono già stati raggiunti dalla tecnologia O.N.D.E.-UWC[1]. Attualmente è possibile effettuare diverse azioni tramite l'app, come una consultazione della mappa per visualizzare la localizzazione dei mezzi di raccolta nel territorio, gestire la propria pagina personale, ecc.

Nonostante sia presente già una versione nativa sullo store, per aggiornarla e migliorarla si è pensato di passare a React Native. I motivi del cambiamento da Java a React Native sono vari e differenti fra loro. Innanzitutto Cidiu4u era stata sviluppata solamente in Java per Android e perciò era necessario sopperire alla mancanza della versione in Objective-C o Swift per iOS, dato che i telefoni Apple al giorno d'oggi ricoprono una grossa fetta di mercato per quanto riguarda gli utenti.

[1]Grazie alla localizzazione dei mezzi e alla pesatura dei contenitori in fase di raccolta, una razionalizzazione dei cicli di raccolta dei rifiuti solidi urbani e della carta sui territori serviti

Figura 4.1: Icona applicazione CIDIU4U

Oltre allo sviluppo di una versione per iOS era importante anche l'aggiornamento dell'applicazione già esistente con un inserimento di nuove funzionalità (come il calendario) e una correzione dei bug esistenti. Dovendo perciò riscrivere l'applicazione sia in Java che in Objective-C, la scelta si è direzionata su React Native per poter scrivere un unico codice e risolvere entrambi i problemi grazie alla sua proprietà di linguaggio multipiattaforma.

4.2 Quale bisogno soddisfa

Cidiu4u è stata sviluppata con l'intento di migliorare la qualità del servizio di raccolta e la qualità di vita dei cittadini, e per soddisfare questi bisogni è necessario arrivare a più utenti possibile. Da qui la scelta di iniziare a sviluppare con React Native per poter raggiungere sia utenti in possesso di dispositivi Android che utenti in possesso di dispositivi Apple. Sviluppare una applicazione in React Native, in questo caso anche un prodotto, è un punto a favore, oltre che essere disponibile per tutti i sistemi operativi, React Native è un linguaggio sempre più usato e improntato al futuro.

Per le caratteristiche e funzionalità che Cidiu4u offre, React Native è consigliato dato che non richiede grosso carico di prestazioni, è un'applicazione di visualizzazione dati principalmente. Necessità di alcune feature messe a disposizione dai vari sistemi operativi, come gps e fotocamera, ma essendo le feature "base" di ogni applicativo, sono supportate comunque da React Native e perciò non si presentano problemi.

Il problema principale che ha creato l'esigenza di apportare modifiche all'applicazione già sviluppata però, è dovuto ad alcuni bug o problemi creatosi nel tempo con vari aggiornamenti ai sistemi operativi o librerie inserite nel progetto andate in disuso, utilizzate per massimizzare il risultato con un impiego di tempo minore. Dovendo perciò modificare l'applicazione precedente e creare una versione per iOS in tempi brevi, React Native è risultato ancora una volta il linguaggio più adatto a questa problematica. Oltre all'aggiornamento di Cidiu4u è previsto un inserimento di nuove funzionalità in aggiunta a quelle già esistenti per migliorare il servizio offerto all'utente.

4.3 Sviluppo dell'app

Cidiu4u ha come funzionalità principale il supporto al cittadino per quanto riguarda la raccolta rifiuti nel suo paese. Questo si traduce come richieste personalizzate HTTP tramite un protocollo sicuro al database personale dell'azienda, la quale si occupa di gestire e mantenere i dati relativi a posizioni, conferimenti generale e personali del cittadino. Una volta ottenuti i dati richiesti, essi necessitano solamente di essere interpretati e visualizzati a schermo. Queste sono le due funzionalità principali che l'applicazione sviluppata offre. L'app ha una pagina principale chiamata HomeScreen, dal quale si possono intraprendere sette differenti percorsi a seconda delle richieste, li descriverò ad uno ad uno in seguito.

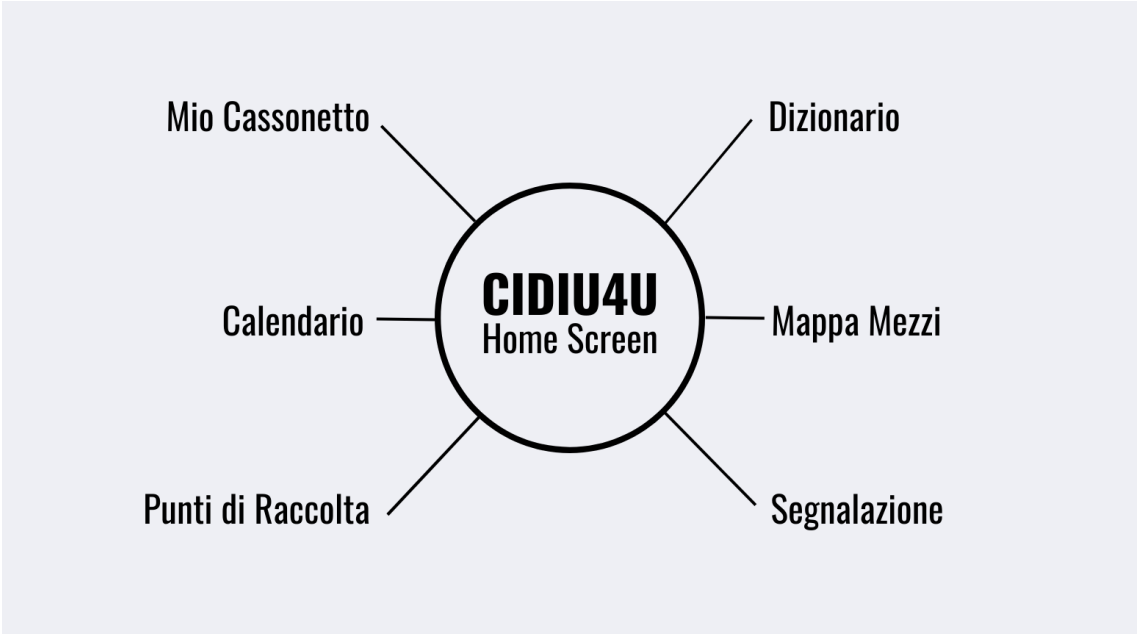

Figura 4.2: Schema concettuale dell'applicazione Cidiu4u

Come collegamento fra le varie activity anche in questo caso si è utilizzata la libreria *react-navigation* con l'activity principale, HomeScreen, è sempre presente nello stack e le activity secondarie che entrano a far parte dello stack una volta eseguito il loro compito vi riescono. Come nell'applicazione precedente, si utilizza NPM come manager dei

pacchetti, il quale si incarica dell'installazione e del link della libreria nel progetto.

Cidiu4u avendo come obbiettivo il raggiungimento del totale dell'utenza, ha la necessità di essere provata e testata anche su iOS, e non disponendo di un Mac [2], si è utilizzato Expo per l'avvio dell'applicazione su iOS. Expo è un insieme di strumenti sviluppati intorno a React Native e che permettono l'avvio dell'app utilizzando un client installato sul dispositivo finale, bypassando così un debug usb ma passando il codice scritto in React Native tramite la rete locale. In Android non è necessario dato che è sufficiente avviare il debugger USB direttamente dal telefono sul quale vogliamo installare l'app, mentre in iOS sono necessari ulteriori permessi e Expo è il metodo più veloce e che si avvicina di più al metodo tradizionale.

Nelle sezioni successive si analizzano le varie activity per composizione e funzionalità.

[2]E' obbligatorio per sviluppo in React Native su Apple

4.3.1 Home Screen

E' la pagina principale dell'applicazione, da qui è possibile spostarsi sulle altre differenti activity rimanendo però sempre attiva sullo stack di navigazione. HomeScreen è composta da alcune immagini rappresentative delle aziende che vi collaborano e da una griglia. La griglia è composta dalla libreria *react-native-super-grid* e in ogni casella di essa è presente un'icona [3] rappresentativa dell'activity sul quale ci si sposterà una volta cliccata la casella. Le celle sono sei come le activity e sono disposte in due colonne e tre righe. Essendo React Native adattivo, le dimensioni delle celle si adattano allo schermo del dispositivo e quindi non sono fisse, questo per poter permettere un utilizzo dell'applicazione su dispositivi con schermi differenti oltre che sistemi operativi diversi. La realizzazione di elementi con dimensioni relative è effettuata grazie a un uso relativo e non assoluto di CSS (fogli di stile che possono essere inclusi nel progetto). Tramite una variabile settata nelle impostazioni, (non implementata) è possibile cambiare la vista dell'HomeScreen: passare da una griglia a due colonne e tre righe, il tutto contenuto in una schermata, a una griglia con un'unica colonna e sei righe, per visualizzare tutte le celle è necessario una ScrollView, le celle saranno più grandi e conterranno più informazioni. E' presente in alto a destra una icona per spostarsi nella sezione delle impostazioni, nel quale in un futuro sarà possibile modificare l'ultima opzione vista.

Figura 4.3: Schermata Home Screen

[3] si utilizza nuovamente la libreria react-native-vector-icons

4.3.2 Mio Cassonetto Screen

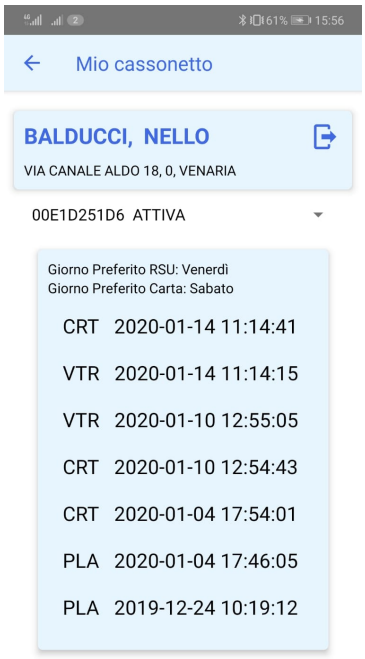

Figura 4.4: Schermata Mio Cassonetto

Questa activity è utile per gli utenti che utilizzano i contenitori interrati con i badge dato che possono visualizzare i propri ultimi conferimenti, ed è differente per ogni utente che la utilizza con dati personali relativi a codice fiscale e via di residenza. L'activity è divisa in due viste, la prima contiene un Cardview (creata con la libreria *react-native-cardview*) con la possibilità di inserire il proprio codice fiscale. Una volta inserito il codice fiscale tramite un HTTP request l'applicazione controllerà che l'utente sia effettivamente registrato nel registro di Cidiu. Se questo avviene si passa alla seconda vista la quale presenta un'altra Cardview contenente nome, cognome e via di residenza. Oltre a queste informazioni ottenute dalla HTTP request si ottengono le diverse carte di conferimento (badge) associate all'utente e vengono inserite all'interno di un picker, creato con la libreria standard di React Native, utile per rendere una ulteriore Cardview dinamica a seconda della carta selezionata. Tramite la carta selezionata è possibile visualizzare se è attiva o meno e quali sono i giorni preferiti per ogni tipologia di conferimento (carta, rsu, plastica, ecc.). Se sono stati effettuati dei conferimenti per tipo, il giorno preferito sarà visibile e saranno visibili inoltre anche un elenco di item, tutti i conferimenti effettuati fino ad oggi.

Utilizzando delle HTTP request, c'è la necessità di dover attendere alcuni secondi prima di ottenere il risultato, nell'attesa è presente un' Activity Indicator (differente a seconda dei sistemi operativi) che indica un caricamento.

L'applicazione gestisce informazioni personali, ma ogni utente che la utilizza dovrebbe ogni volta inserire nuovamente il codice fiscale e

per evitare ciò, entra in gioco un salvataggio dei dati come variabile globale e un salvataggio come risorsa in locale tramite la libreria *@react-native-community/async-storage*. L'utente non dovrà più inserire queste informazioni ogni volta che avvia l'applicazione oppure ogni volta che cambia activity, perchè il codice fiscale verrà salvato all'esterno dell'applicazione e ogni volta che si aprirà l'activity sarà immediatamente posizionata sulla seconda vista. Nonostante ciò se si vuole cambiare utente o comunque disconnettersi, è presente l'icona logout che permette di cancellare il codice fiscale precedentemente salvato e riposizionarsi con la vista incaricata di leggere un nuovo codice fiscale.

4.3.3 Dizionario Screen

Dizionario Screen è un'altra activity che si basa su azioni HTTP request e visualizzazione dati, ma questa volta effettua una richiesta HTTP intermedia. Come impostazione risulta simile alla precedente perchè al suo avvio si presenta una CardView con una TextInput al suo interno con la possibilità di inserire del testo. Una volta inseriti almeno tre caratteri diventa visibile una lista di possibili risultati di ricerca che contengono nel risultato il testo della TextInput, creando così una sorta di suggerimento alla ricerca. Ad ogni carattere inserito questa funzione di suggerimenti si ripete, aiutando l'utente nella ricerca. Ottenuto il suggerimento che si desiderava è possibile cliccarlo, cosa che attiva un'altra funzione di ricerca e che restituisce importanti informazioni sul rifiuto in questione. Le informazioni vengono catalogate in tre differenti categorie e rappresentate ognuna in una Cardview differente, esse sono destinazione, tipologie e note, quest'ultima è opzionale quindi a seconda della sua presenza la

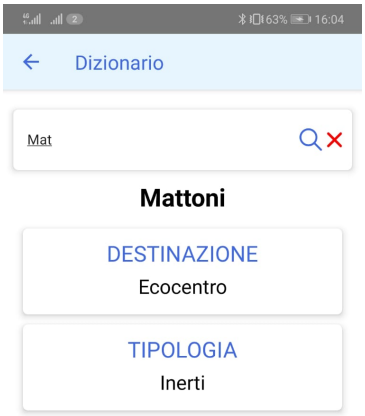

Figura 4.5: Schermata Dizionario

cardview risulterà visibile o meno. In qualsiasi momento è possibile cancellare il testo inserito e le CardView premendo il tasto rappresentato da una "X"[4]. E' presente anche un'icona per la funzione di ricerca ma che svolge le stesse funzioni del click su un determinato suggerimento.

4.3.4 Calendario Screen

Figura 4.6: Schermata selezione via in Calendario

E' l'activity che rappresenta la principale funzionalità inserita dopo le modifiche, per quanto riguarda il funzionamento esso si vedrà nella sezione successiva, in questa sezione si analizza da quale parti è composto. E' composta da quattro differenti viste tutte creata grazie a if condizionali nella sezione di render() che ci permette di ottenere diverse sezioni di codice partendo da alcune variabili stato mutevoli nel tempo. Le variabili indicate sono: *citta_bool, via_bool, civic_bool, result*, esse sono inizializzate a 0 ad eccezione di *citta_bool* che assume il valore 1. A seconda di chi ha il valore 1 avremo una vista differente e ogni volta che il compito da svolgere sarà completato, la variabile settata a 1 tornerà 0 e assegnerà l'1 ad una delle tre restanti, passando il testimone.

Oltre alla logica delle viste, è presente una fase di inizializzazione necessaria per ottenere i parametri di accesso alle API incaricate di gestire le HTTP request per quanto riguarda comuni, vie e calendario giornaliero. E' preimpostato un payload contenente alcune variabili come 'user', 'apikey' e 'hash'[5], tramite esso è

[4]E' costituita da un'icona colorata di rosso, che sta ad indicare l'azione 'cancella'

[5]Hash calcolato grazie a una libreria che effettua una traduzione in md5, la libreria si chiama semplicemente *md5*

possibile effettuare una sorta di autenticazione lato server effettuando un POST [6] con il payload all'url impostato. Si utilizza un protocollo per la comunicazione sicura per tutte le richieste, quindi HTTPS. In risposta alla richiesta di accesso si otterranno due variabili, id e token, necessari per effettuare l'accesso e verificare l'effettiva identità dell'utente per tutte le richieste e quindi POST successivi.

Le viste hanno una logica lineare, è possibile passare dalla prima alla seconda, dalla seconda alla terza e dalla terza alla quarta, per poi eventualmente tornare indietro in qualunque momento o all'inizio. La prima vista è composta da una FlatList contente tutti i comuni gestiti da una raccolta porta-a-porta, una richiesta HTTP è incaricata di riempire questa lista. Una volta selezionata la città è possibile cambiando il valore delle variabili stato passare alla seconda vista incaricata di ottenere il nome della via. Il funzionamento è simile a quello della TextInput del Dizionario Screen, ovvero si attendono almeno 3 caratteri per poter visualizzare a schermo una serie di suggerimenti di vie appartenenti al comune selezionato in precedenza che contengono quel testo. E' presente una "X" per poter tornare indietro nel caso in cui la selezione del comune fosse sbagliata[7]. Selezionata la via si passa alla terza vista, dove sarà possibile, tramite un TextInput numerico, inserire il numero civico, anche in questo caso è possibile tornare alla vista precedente tramite un icona.

Figura 4.7: Schermata finale Calendario

Selezionato anche l'ultimo dato, si hanno a disposizione tutti i dati

[6]Il POST viene utilizzato per inviare dati a un server per creare/aggiornare una risorsa

[7]La prima immagine della sezione rappresenta questa acctivity

per completare il payload per ottenere la raccolta differenziata giornaliera, è necessario solamente sapere il giorno. Dovendo formare un calendario, si ha bisogno di sapere la data esatta per ogni giorno della settimana corrente, da lunedì a domenica. E' possibile ottenerlo grazie alla funzione Date()[8] la quale restituisce il momento esatto corretto, dopo alcuni calcoli da parte di una funzione sarà presente una lista completa dei giorni. Con i dati precedenti e i giorni della settimana è possibile ottenere quale tipo di raccolta porta a porta viene effettuata per una determinato luogo (identificato da civico, via e comune) in un determinato giorno.

Nell'ultima vista (caratterizzata dalla variabile *result* settata a 1) avremo un titolo che evidenzia le variabili raccolte in precedenza (civico, via e comune) e un sottotitolo che indica la settimana presa in considerazione, inizialmente è la settimana corrente, con la possibilità di poter passare alla settimana successiva o precedente. Sono presenti sette CardView, una per ogni giorno della settimana, e all'interno di esse sono rappresentati i dati ottenuti, tipo di raccolta differenziata per quel giorno e orario di raccolta. Il giorno corrente è rappresentato scritto in grassetto.

Come nelle viste precedenti è presente un'icona per ripetere il processo e perciò cancellare i risultati ottenuti e tornare alla prima vista per ri-effettuare una scelta del comune e tutte le scelte successive.

[8]E' una funzione principale in Javascript che restituisce l'esatta ora e data del sistema operativo in funzione.

4.3.5 Mappa Mezzi Screen

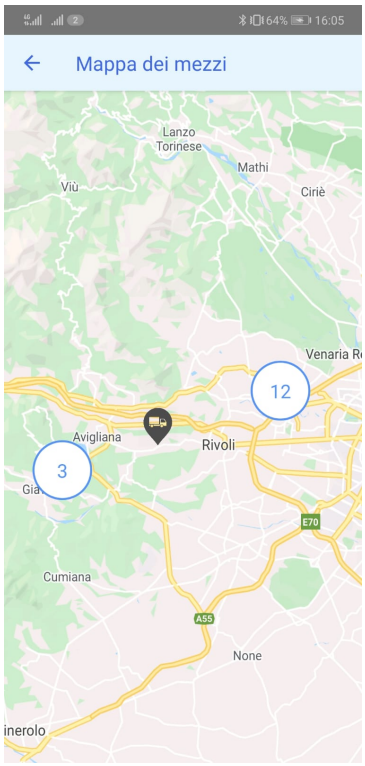

Figura 4.8: Schermata Mappa Mezzi

Questa activity presenta una funzionalità presente già nella versione precedente ma aggiornata. E' presente nella totalità dell'activity una mappa intenta a rappresentare i mezzi di raccolta rifiuti sparsi per il territorio, rappresentati da un insieme di marker. Per differenziarli da un normale punto di localizzazione sulla mappa, sono state personalizzate le icone dei marker sulla mappa[9] con icone rappresentanti un camion dei rifiuti. Si è migliorata la rappresentazione dei punti di localizzazione sulla mappa inserendo una mappa ottenuta da una libreria differente *react-native-cluster-map*, la quale permette di creare dei cluster di marker ovvero raggruppare i punti di localizzazione a seconda dello zoom presente sulla mappa. Mano a mano che si aumenterà lo zoom anche i cluster si divideranno in cluster più piccoli, fino ad essere rappresentati dal singolo marker. Essendo la mappa e i relativi marker un carico impegnativo, è necessario impostare un'activity indicator per segnalare la presenza di un caricamento all'utente. I dati relativi alle posizioni dei mezzi sono aggiornate ogni 5 minuti.

[9]Icona dei marker personalizzati salvata in locale, motivo del maggior tempo di caricamento

4.3.6 Punti Raccolta Screen

La schermata Punti di Raccolta si presenta molto simile all'activity descritta in precedenza, dato che è composta principalmente da una mappa, anch'essa ottenuta dalla libreria *react-native-cluster-map*, con una visualizzazione a cluster ma con alcune funzionalità differenti. Innanzitutto i punti di localizzazione sulla mappa sono più numerosi, all'incirca 1500, e rappresentano i siti di raccolta presenti nei vari comuni serviti da Cidiu SpA. Ogni sito può contenere bidoni di cinque diverse tipologie di rifiuto: tessile, carta, plastica, RSU e vetro. Per ognuno di essi è stato creato un filtro, posizionato in alto sopra a mappa, utile per filtrare i siti di raccolta che contengono il rifiuto del filtro selezionato. A seconda del filtro selezionato i marker presenti sulla mappa diminuiranno o aumenteranno e i cluster formati saranno differenti. I siti di raccolta sono ottenuti tramite una richiesta al database di appoggio con una semplice HTTP request, necessaria per salvare i dati su una lista. Questa operazione viene effettuata soltanto la prima volta di modo da non dover caricare l'applicazione di eccessive richieste HTTP.

Figura 4.9: Schermata mappa Punti Raccolta

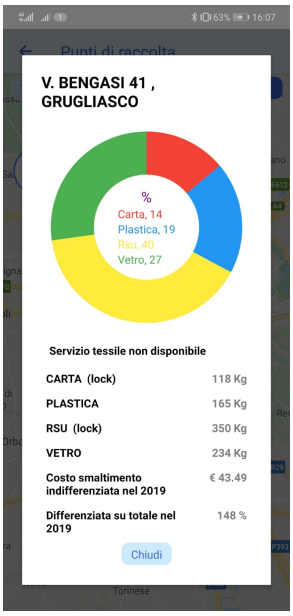

Figura 4.10: Schermata torta Punti Raccolta

I vari siti, rappresentati dai marker, sono cliccabili e una volta cliccati si possono ottenere alcune statistiche annuali relative al sito premuto. Ogni marker ha

un id_sito relativo ad esso, utile per reperire le informazioni e statistiche dal database. Premuto un marker, diventerà visibile un dialog[10] con al suo interno le statistiche del sito corrente. Esse sono rappresentate tramite un grafico a torta, creato con la libreria *react-native-pie-chart*, e altre mostrati a schermo. Questi dati non sono modificabili ma solamente visibili da parte dell'utente. Una volta terminata la ricerca è possibile chiudere il dialog e successivamente, dopo aver cliccato su un altro marker, aprirne un altro.

4.3.7 Segnalazione Screen

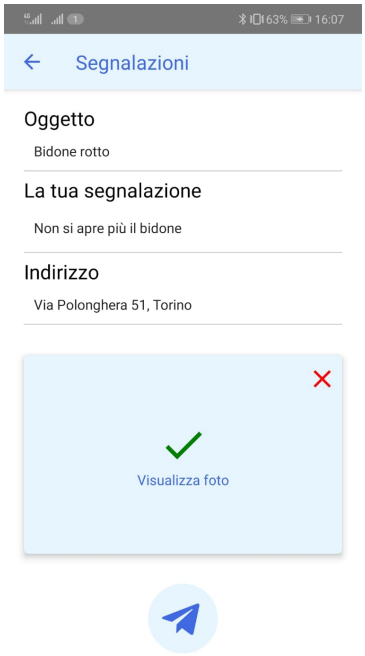

Figura 4.11: Schermata Schermata segnalazione

L'activity Segnalazione è l'activity con più interattività e permette di svolgere più funzioni all'utente anche grazie all'uso di svariate librerie esterne. Innanzitutto sono presenti tre TextInput utili per completare la mail finale e quindi la segnalazione. Per quanto riguarda l'oggetto e la segnalazione sono banali campi di testo che l'utente dovrà completare, mentre l'indirizzo si autocompleterà con l'indirizzo relativo alla posizione dell'utente in quell'istante ricavato dalla geolocalizzazione del telefono cellulare. Il nome della via è ottenuto nuovamente dalla libreria *nominatim-geocoder*, ma essendo possibile un invio di segnalazione anche non nell'immediato, la TextInput è modificabile inserendo una qualsiasi via. In seguito è presente una CardView che esegue una funzione incaricata di ottenere un uri relativo a una foto del danno protagonista della segnalazione. Cliccando sulla Cardview si aprirà un Dialog incaricato di porre una scelta all'utente, se reperire la foto dalla

[10]Dialog creato grazie alla libreria *react-native-simple-dialogs*

fotocamera o dalla galleria, se la foto è stata scattata precedentemente. Queste funzioni sono realizzabili grazie alla libreria *react-native-image-picker* la quale a seconda della scelta va a prendere l'uri relativo alla foto selezionata o scattata. Una volta effettuato questo procedimento la foto risulterà salvata e sarà possibile cancellarla e rivederla in ogni momento [11]. Completati tutti i campi e ottenuto l'uri della foto, premendo il bottone invia, è possibile creare un template per l'invio di una mail con il proprio indirizzo mail. La scelta è fra alcune applicazioni di invio mail disponibili sul telefono personale ma saranno completate grazie ai parametri inseriti, avranno come oggetto e corpo della mail i vari TextInput completati e come allegato la foto selezionata.

[11]Si utilizza la libreria *react-native-image-picker* per visualizzare la foto a schermo

4.4 Funzionamento dell'app

L'applicazione Cidiu4u è un'app che ha come principale funzione il supporto al cittadino per quanto riguarda la raccolta dei rifiuti nei comuni nel torinese serviti da Cidiu SpA. Ogni activity presente ha funzionalità e regole differenti, alcune ereditate dalla versione precedente altre o modificate o create da zero. Al primo impatto la modifica principale che si può notare è un differente stile utilizzato, sia per il posizionamento dei componenti all'interno delle varie activity, sia per quanto riguarda il colore dei vari elementi, questo per mostrare al primo impatto che sono state effettuate alcune modifiche e per allontanarsi dai colori standard che Android offre.

Per il funzionamento, Cidiu4u permette un controllo del lavoro svolto da altri, vedi l'activity mappa mezzi, un controllo su lavoro passato, vedi l'activity Punti di raccolta e un controllo e gestione da parte dell'utente che può verificare i suoi passati conferimenti, capire come e quando effettuare i conferimenti futuri grazie al calendario e al dizionario e inoltre segnalare eventuali problemi nel processo attraverso la pagina di segnalazione.

La Mappa dei mezzi di raccolta rifiuti è aggiornata in tempo reale, con un aggiornamento ogni 5 minuti del database grazie ai mezzi in transito nelle città dotati di localizzazione satellitare, e permette all'utente di informarsi sulla posizione in tempo reale per evitare eventuali ingorghi del traffico creatosi per lo smaltimento dei siti in quella posizione, risultando così un'applicazione utile nell'immediato.

Cidiu4u consente ai cittadini di visualizzare i conferimenti nell'anno passato e l'effettivo costo di smaltimento dell'indifferenziata, per ogni sito, attraverso l'activity Punti di raccolta, è possibile visionare il totale dei chili raccolti per tipo durante l'anno, visualizzati all'interno di un dialog con un valore percentuale che andrà a comporre un grafico a torta. Oltre a questi dati reperiti dal database è possibile informarsi sull'eventuale chiusura del cassonetto e una percentuale della differenziata sulla raccolta totale nel 2019. Con questa funzione è possibile informarsi sulle raccolte degli anni passati, ma nell'immediato è necessaria per sapere l'effettiva posizione di un sito se c'è una necessità da parte del cittadino di buttare dei rifiuti.

Insieme al dove buttare un rifiuto è necessario da parte del cittadino sapere il come buttare il rifiuto, e qui entra in gioco l'activity Dizionario, attraverso il quale un utente può, partendo dal tipo di rifiuto che deve buttare, ottenere informazioni come destinazione, avvertimenti e consigli (come nel caso del tetrapack la divisione dei materiali differenti). Ogni cosa può essere considerata un rifiuto e per non accettare richieste non gestite dal database è inserita una funzione dei suggerimenti che partendo da una stringa in ingresso restituisce i vari tipi di rifiuti presenti nel database e che corrispondono alla stringa.

Le funzionalità descritte gestiscono un conferimento di rifiuti nei bidoni posti a bordo strada, ma per i bidoni interessati al porta-a-porta sono presenti ulteriori funzionalità, una delle quali è il Calendario, che consiste nel visualizzare una settimana scelta (corrente e non) nel quale giorno per giorno, viene indicato quale rifiuto gli addetti alla raccolta passeranno a prendere e in che fascia oraria, per permettere al cittadino di preparare il suo rifiuto per quel giorno e ora. Ovviamente il porta-a-porta interessa in un modo alcune città e in un altro altrettante città, ed è perciò obbligatorio l'inserimento di città, via e civico nell'activity Calendario. Sono presenti anche qui i suggerimenti per evitare un inserimento di città o vie errate.

Per visualizzare i conferimenti passati e tenere traccia di quanto e quando un cittadino ha utilizzato il proprio porta-a-porta, esclusivamente per quei cittadini con i contenitori interrati e provvisti di badge, è presente l'activity Mio Cassonetto nel quale trovano posto tutte le tessere personali dell'utente. Riguardando un caso personale, questa è un'activity differente ad ogni utente al quale vi si può accedere solamente l'utente interessato.

Per il corretto funzionamento di tutte le attività è necessario che gli oggetti materiali quindi i bidoni siano corretti e funzionanti in ogni momento, se questo non dovesse accadere entra in gioco l'ultima activity, l'activity di segnalazione che permette di avvertire l'amministrazione incaricata di gestire il corretto funzionamento dei bidoni, della presenza di un problema. E' possibile avvertire mostrando con una foto il problema e/o scrivendo alcune righe di testo per spiegare la problematica, fatto ciò è necessario mandare una mail con il proprio indirizzo personale, di modo da poter identificare l'ideatore della segnalazione

per eventuali approfondimenti.

Cidiu4u è stata pensata per un uso in caso di necessità e non per un uso di intrattenimento e perciò la vista in landscape non è supportata. Ha bisogno di alcuni permessi da inserire nel manifest [12] come l'accesso a internet, fotocamera, geolocalizzazione e memoria interna. Essendo un'applicazione basata sull'ottenere dati da un database o inviarli allo stesso database, un utilizzo senza connessione dati non porterebbe al corretto funzionamento della stessa.

L'utilizzo di React Native permette di avere aggiornamenti veloci e continui del codice, e quindi minor tempo di sviluppo. Essendo un'app destinata al cittadino, React Native diventa necessario per raggiungere la quasi totalità delle utenze ricoprendo l'esigenza sia di Android che di iOS.

[12]Il Manifest raccoglie informazioni basilari sull'app, informazioni necessarie al sistema per far girare qualsiasi porzione di codice della stessa.

Capitolo 5

Best practices

Dopo essermi informato sulle caratteristiche, struttura e architettura di React Native, dopo un confronto con altri linguaggi simili e non, React Native è risultato adatto al tipo di applicazione che si desiderava sviluppare con la creazione dei due progetti Tag Reader App e Cidiu4u. Una volta completate le due applicazioni e quindi aver passato del tempo a lavorare con React Native, posso affermare di aver assimilato alcune Best Practice, utili e/o necessarie per lo svliuppo di un applicativo in React Native.

Per Best Practice si intendono perciò delle linee guida ricavabili dall'esperienza appena avvenuta, che ha permesso di ottenere dei buoni risultati sviluppando due applicativi in React Native.

Per elencarle, le dividerò in due gruppi, "cose da fare" e "cose da non fare".

5.1 Cose da fare

Come in un qualsiasi progetto, anche non in un contesto informatico, l'ordine e l'organizzazione sono fattori essenziali per lo sviluppo dello stesso. La logica di organizzazione di un progetto sviluppato in React Native, non si discosta molto da una logica di un progetto sviluppato in React, e consiste nella suddivisione dei diversi elementi presenti nel progetto in cartelle differenti come: *assets*, *components*, *views*, *services*, *configuration* e altre a seconda delle esigenze. Ognuna di esse contiene alcuni file suddivisi per ruolo all'interno del progetto, *assets* contiene icone, immagini e fonts, *components* ha al suo interno tutti gli elementi privi di una logica, *views* contiene i vari screen presenti all'interno dell'applicazione, mentre *services* e *configuration* presentano al loro interno API di servizio e file di configurazione. Ordinare tutti i file in cartelle suddivise per funzione permette, una volta localizzata la fonte dell'errore di poterlo identificare e risolvere facilmente.

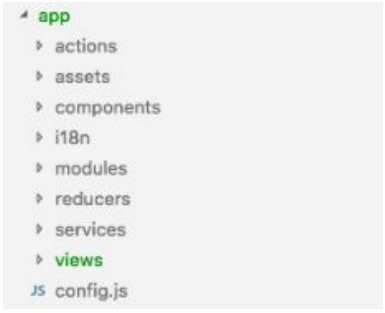

Figura 5.1: Organizzazione in cartelle.

Un'organizzazione così pensata risulta efficiente, soprattutto in termine di riusabilità se i componenti creati sono il più basilare e semplice possibile (come container, header, button, ecc.). Con più componenti semplici, si riduce il numero di righe di codice ripetute ma si ripeteranno solo le chiamate a quello specifico componente. Si ridurrà al minimo anche il lavoro impiegato per riscrivere lo stesso codice un'altra volta, si dovrebbe far fede alla regola DRY (*"Don't Repeat Yourself"*) la quale spiega che se c'è del codice ripetuto è possibile ridurlo. I componenti creati, essendo più semplici e riusabili semplificano la fase di test e

debug, ma dovranno necessariamente essere senza stato per essere utilizzati in contesti differenti. Inoltre se essi svolgono bene la funzione per la quale sono stati sviluppati, possono anche essere messi a disposizione della community tramite alcuni siti di supporto come Github. Come i componenti anche lo stile è necessario che sia riusabile e per questo motivo viene raggruppato in un file, solitamente *style.js*, incluso dai vari componenti e screen, di modo da essere comune a tutti ma scritto solamente una volta.

Per quanto riguarda lo stile, è necessario che sia al più simile ad ogni piattaforma e dispositivo ed è possibile solamente utilizzando le dimensioni relative in CSS e non assolute[1]. Per gestire al meglio la questione delle dimensioni relative, React Native mette a disposizione una API utile ad ottenere la dimensione dello schermo utilizzato in quel momento dall'applicazione, essa si chiama *Dimensions()*. Se non si vuole utilizzare la API messa a disposzione da React Native è anche possibile ottenere lo stesso risultato con librerie create da terze parti.

Dato che è importante ridurre al minimo il numero di codice utilizzato è possibile e consigliabile anche creare un file .js con al suo interno tutte le varie inclusiononi di screen, componenti, ma soprattutto di librerie esterne, di modo da evitare di doverli ripetere ogni volta e rischiare di dimenticarsi creando così alcuni errori. E' presente però un momento in cui è necessario scrivere più volte una porzione di codice, e questo avviene quando l'applicazione deve risultare differente per due piattaforme differenti come Android e iOS. Viene utilizzata un'altra API fornita dalle librerie caratteristiche di React Native e si chiama *Platform()*. Tramite un semplice if condizionale è possibile differenziare il codice a seconda se l'applicazione è avviata su un dispositivo Android o un dispositivo iOS. Un esempio per cui questo caso viene applicato è l'utilizzo dell'Activity Indicator[2], è necessario usare una versione diversa a seconda del sistema operativo utilizzato.

La libreria più usata è importante in questo campo, è la libreria di

[1]Le dimensioni relative sono rappresentate da un numero in percentuale, mentre quelle assolute da un numero fisso

[2]Componente responsabile di visualizzare schermata di caricamento quando i dati non sono ancora stati caricati

navigazione dato che mette in comunicazione diversi screen dell'applicazione e permette di muoversi all'interno di essa. E' necessario perciò avere una libreria che sia comoda e veloce. Dato che non è standard ma possibile solamente utilizzare una libreria creata da terzi. Si possono usare due diversi approcci, uno javascript e uno nativo, ma nel caso dei due progetti sviluppati si utilizza per entrambi la libreria *react-navigation* con il supporto di *react-navigation-stack* che utilizza un approccio con javascript.

In fase di sviluppo del codice è necessario pensare specialmente alle funzionalità dell'applicazioni che devono poter essere eseguite indipendentemente dal loro ordine e organizzazione. Riuscire a non interrompersi in un determinato screen, far sapere all'utente in ogni momento cosa succede premendo un bottone o eseguendo una specifica gesture, poter interrompere un'azione a metà, rendono l'utente a suo agio e consentono ad esso di poter provare e sperimentare l'applicazione senza aver paura di sbagliare. Il fatto che un utente si trovi a suo agio con l'utilizzo dell'applicazione è un parametro estremamente importante per la vendita e l'installazione dell'app su i vari dispositivi.

Una volta terminata la fase di scrittura e sviluppo del codice, è di estrema importanza la fase di test e debug, e per poter diminuire questo controllo nella fase finale è possibile usare una libreria, Redux, incaricata di gestire una corretta comunicazione tra componenti e gestire lo stato corrente. Il suo scopo è quello di rendere prevedibile il cambio di stato tramite l'imposizione di alcune restrizioni su come e quando questo può avvenire e perciò diminuire i problemi che si possono creare in fase di test e debug. Per quanto riguarda la fase di test invece si utilizza un altro framework, usato specialemente per le applicazioni Javascript, chiamato Jest[3], dato che l'importanza di essi è oramai consolidata e ogni azienda matura ne fa un largo uso per garantire l'affidabilità delle soluzioni realizzate.

[3]Simile ai test che si effettuano in Java normalmente

5.2 Cose da non fare

Oltre ai consigli e alle cose da fare in un progetto, ho constatato che ci sono anche cose da non fare per la buona riuscita di un progetto, alcune riconducibili ai consigli elencati in precedenza. In ogni progetto è buona cosa commentare il codice per tenere traccia di cosa si è fatto e per aiutare un futuro sviluppatore che dovrà leggere il proprio codice, ma anche la quantità dei commenti è importante, troppi commenti rischiano di confondere la presenza di commenti per informare ciò che fa una funzione o commenti creati in fase di debug per testare le funzionalità di un'applicazione.

Come i commenti anche i nomi che uno sviluppatore sceglie per variabili e funzioni sono estremamente importanti per capire subito quale azione svolge una determinata funzione, è perciò importante scrivere nomi sintetici ma che riassumano il loro contenuto e non nomi solitamente generici che vengono dimenticati a distanza di qualche settimana dallo sviluppo. Ad esempio per una funzione che cerca un nome all'interno di una lista è conveniente chiamarla *func_cerc_nome* piuttosto che *funzione1*. Stesso discorso è valido per le variabili, a differenza che qua il riassunto si basa su cosa contengono e non su cosa fanno. Sempre per quanto riguarda l'uso dei nomi degli elementi è importante differenziare variabili e funzioni in partenza, assegnando tutti caratteri minuscoli per le variabili e carattere iniziale maiuscolo per le funzioni.

Le librerie esterne sono fondamentali, possono velocizzare il processo di sviluppo, importando qualcosa che è già stato fatto, o migliorarlo andando ad aggiungere funzioni per cui non si ha tempo o capacità per svilupparle. In ogni progetto è possibile includere una quantità infinità di librerie esterne ed alcune sono già incluse in fase di creazione del progetto. Ma ogni libreria presenta una versione e ogni libreria è sempre in attesa di essere aggiornata, l'aggiornamento è utile in quanto corregge eventuali errori presenti e/o migliora il sistema offerto dalla libreria, ma aggiornandosi, può creare alcune incompatibilità con il codice già sviluppato che non erano presenti prima dell'aggiornamento. Queste incompatibilità possono portare al malfunzionamento dell'applicazione ed essendo presenti molti aggiornamenti di qualsiasi tipo (da React Native al sistema operativo) è molto probabile che si verifichino.

Per evitare che ciò accada è necessario bloccare le dipendenze a quella libreria a una versione stabile e testata, che funzioni con l'ambiente circostante.

In termini di sviluppo è importante prestare attenzione sui fogli di stile CSS, in quanto alcune scorciatoie di determinate proprietà non sono state implementate e perciò risultano malfunzionanti, un esempio è l'uso della percentuale in alcune proprietà.

Grazie alle best-practice scritte in precedenza, è possibile massimizzare il lavoro diminuendo il tempo impiegato nello sviluppo del progetto, alcune di esse non sono proprie di React Native ma di una programmazione in generale, ma nel complesso per la creazione di un'applicazione sono utili tutte.

Conclusioni

In questo lavoro di tesi, l'obbiettivo principale era quello dell'impadronimento di una nuova tecnica di sviluppo mobile multipiattaforma che sta prendendo piede negli ultimi anni. Dopo una semplice ricerca sul perchè è nata e sul perchè questo framework è più adatto in alcune situazioni rispetto ad altri, se ne sono calcolati i pro e i contro di modo da aver sufficienti fattori per effettuare una scelta.

Una volta effettuata la ricerca teorica, ho potuto impratichirmi su React Native tramite lo sviluppo di due applicativi in collaborazione con MoltoSenso srl e Cidiu S.P.A. Entrambe le applicazioni sono sviluppate interamente in React Native e sono a stretto contatto con una tecnologia già esistente incaricata per la raccolta differenziata e non dei rifiuti in periferia torinese. Le applicazioni sono però a tutti gli effetti dei prodotti utilizzabili e non soltanto dei progetti scolastici.

Grazie allo sviluppo delle app in React Native ho potuto assimilare delle Best Practice riguardanti lo sviluppo di un applicativo in React Native e il suo settaggio. Insieme ai vari pro e contro del framework e le varie differenze rispetto a framework simili trattati nella tesi, forniscono un buon punto di partenza per poter capire se React Native può essere o meno una scelta adatta per lo sviluppo di un nuovo progetto innovativo.

Bibliografia

[1] Bonnie Eisenman, *Learning React Native*, O' REILLY, Dicembre 2015

[2] Jonathan Lebensold, *React Native Cookbook: Bringing the Web to Native Platforms*, O' REILLY, 2018

[3] Nader Dabit, *React Native in Action, developing iOS and Android apps with JavaScript*, MANNING PUBBLICATIONS, aprile 2019

[4] Eric Masiello, Jacob Friedmann, *Mastering React Native*, PACKT, gennaio 2017

[5] Ethan Holmes, Tom Bray, Jacob Friedmann, *Getting Started with React Native*, PACKT, Dicembre 2015

[6] React Native
`www.reactnative.dev`

[7] React Native
`https://github.com/facebook/react-native`

[8] Che cos'è React Native
`https://www.quora.com/What-is-React-Native`

[9] Futuro di React Native
`https://brainhub.eu/blog/future-react-native/`

[10] ReactJS
`https://reactjs.org/`

[11] Guida React Native
`https://www.ideamotive.co/react-native-development-guide/`

[12] State & Props
`https://www.opencodez.com/react-native-props-and-state`

[13] Virtual Dom
`https://www.codecademy.com/articles/react-virtual-dom`

[14] Codice nativo
https://www.creact.it/nativo-vs-react-native
[15] Flutter
https://nevercode.io/blog/flutter-vs-react-native
[16] Flutter
https://www.simoncode.net/flutter-vs-react-native
[17] Ionic
https://ionicframework.com/articles/
ionic-vs-react-native
[18] NPM
https://www.npmjs.com/
[19] Infografiche
https://create.piktochart.com/
[20] Best Practices
https://www.innofied.com/top-10-react-native-best-practices