POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Sviluppo di una applicazione multipiattaforma basata sul framework Instant Developer Cloud

Relatore
prof. Guido Albertengo

Laureando
Paolo La Barbera

Tutore Aziendale
Kelyan Spa
Virginio Marengo

Aprile 2020

Alla mia famiglia e ai miei amici

Ringraziamenti

Ringrazio *Virginio Marengo*, il mio tutor aziendale, e *Kelyan S.p.a* tutta, che mi hanno permesso di lavorare nei loro uffici affidandomi un lavoro stimolante ed innovativo, grazie al quale ho potuto proporre e sviluppare le mie idee con flessibilità e autonomia, venendo sempre ascoltato e ricevendo ottime linee guida.
Grazie al professore *Guido Albertengo*, che ha accettato di farmi da relatore per questa tesi, indirizzando il mio lavoro con prezioni consigli, senza i quali non avrei ottenuto lo stesso risultato.
Infine un ringraziamento speciale alla mia famiglia, che mi ha permesso di compiere questo percorso di studi, assistendomi materialmente, ma sopprattuto moralmente, aiutandomi a raggiungere i miei obiettivi.

Sommario

Questa tesi si propone di effettuare una panormica sul mondo delle applicazioni, analizzare i principali approcci alla programmazione multipiattaforma e presentare un caso concreto di sviluppo utilizzando una di queste tecnologie, per dimostrare i vantaggi che essa comporta.
Gli argomenti vengono presentati in due sezioni principali.
La prima occupa i capitoli 2 e 3, e analizza gli aspetti generali legati alle applicazioni: andamento del mercato, sistemi operativi e ambienti di sviluppo, focalizzandosi su quelli multipiattaforma.
In particolare nel capitolo 2 viene presentata una analisi storica che intende analizzare lo sviluppo che ha portato alle app odierne, con dati di mercato e numeri indicativi, legati all'utilizzo, che permettono di avere un quadro generale dell'argomento. In seguito vengono analizzati in principali sistemi operativi, iOS e Android, secondo i loro aspetti principali, e le possiblità di sviluppo che essi offrono.
Nel capitolo 3 vengono analizzati i principali approcci multipiattaforma, come l'approccio Web delle PWA o quello ibrido, e presentati gli aspetti legati alla programmazione in Cloud.
La seconda parte invece racchiude i capitoli 4 e 5. Il loro obiettivo è quello di presentare le tecniche utilizzate per la programmazione con il framework Instant Developer Cloud, scelto per lo sviluppo dell'applicazione multipiattaforma. Il framework si basa sui servizi offerti da *Google Cloud Platform*, per gestire l'infrastruttura, e *su Apache Cordova*, per l'esecuzione dell'app all'interno di ambienti nativi differenti.
L'applicazione sviluppata e presentata nel capitolo 5, è stata eseguita per conto dell'azienda *Kelyan Spa*, nella sede di Savigliano (CN), con l'obiettivo di estendere a più piattaforme uno dei servizi dell'azienda già presente sul mercato, K-4Sales, una soluzione Cloud per l'automazione del processo di vendita, disponibile solamente per iPad.

Indice

Elenco delle figure

Elenco dei listati

Capitolo 1

Introduzione

1.1 Motivazioni

Le applicazioni per smartphone non stanno solamente rivoluzionando il modo in cui utilizziamo i nostri telefoni, ma riescono ad influenzare sempre di più le nostre abitui quotidiane. Esse ci permettono oggi di stare in contatto con persone, acquisire abilità, intrattienerci, effettuare acquisti, semplicemente attraverso l'utilizzo dei dispositivi che abbiamo costantemente a portata di mano.
Questa innovazione tecnologica ha ormai invaso tutti i paesi, compresi quelli in via di sviluppo, come India, Cina, Brasile e Indonesia, che già dal 2017 si collocano in cima alla classifica per numero di app scaricate [1].
Nel 2019, gli utilizzatori di smartphone nel mondo sono stati 2.7 miliardi, con 1.5 miliardi per i tabelt, ed il tempo trascorso su questi dispositivi è per il 90% all'interno delle app [2].
Il mercato delle applicazioni *mobile* infatti non conosce crisi ormai da un decennio e le aziende hanno approfittato di questo trend. Se da un lato sono nate nuove compagnie che sviluppano esclusivamente applicazioni per dispositivi mobili, dall'altro molte aziende informatiche si sono evolute in questa direzione, cercando di implementare i loro software già esistenti all'interno delle app.
E' questo il caso di Kelyan Spa, che negli anni ha sviluppato un servizio per la gestione degli ordini, sottoforma di applicazione per iPad, chiamato K-4Sales. L'azienda mi ha proposto di estendere questo servizio, sviluppando un prototipo di applicazione multipiattaforma eseguibile sia su smartphone che tablet.

La mia curiosità verso il mondo dello sviluppo delle applicazioni, e la possibilità di poter avere una esperienza professionale in una azienda di settore, mi hanno spinto ad accettare questo lavoro e sviluppare per esso questa tesi.

1.2 Obiettivi

L'obiettivo di questa tesi è quello di analizzare il panorama delle applicazioni odierno e approfondire gli aspetti che riguardano lo sviluppo multipiattaforma, applicando una di queste tecnologie in un caso concreto.
Nella prima parte della tesi infatti analizzerò il mercato, i numeri, le tecnologie di sviluppo e gli approcci multipiattaforma legati alle applicazioni per smartphone e tablet. L'obiettivo è quello di presentare i principali sistemi operativi, seguiti dai framework più utilizzati nello sviluppo multipiattaforma, mettendoli a confronto e analizzandone gli aspetti positivi e negativi.
La seconda parte si propone di effettuare una scelta di una delle tecnologie presentate nella prima sezione, in funzione dello sviluppo di una applicazione. Lo scopo principale sarà appunto quello di analizzare gli aspetti rilevanti e sfruttare le caratteristiche del framework scelto, per poi implementarle per lo sviluppo di una applicazione per conto dell'azienda Kelyan S.p.a, nella sede di Savigliano (CN). L'applicazione dovrà rispecchiare la logica di quella attualmente in commercio, ma con una approccio multipiattaforma, per estendere il servizio su più sistemi operativi e più tipologie di device. Sarà quindi compito del lavoro di sviluppo dimostrare i vantaggi che derivano da un approccio multipiattaforma, cercando un riscontro concreto nel mio caso di studio, rispetto agli aspetti teorici discussi nella tesi, durante i primi due capitoli.

Capitolo 2

Il Mondo delle app

2.1 Cenni storici

Come per ogni nuova tecnologia, è opportuno analizzare i passi che ne hanno caratterizzato la diffusione e lo sviluppo, per riuscire a comprenderne meglio gli aspetti odierni e cercare di prevedere le direzioni future.
Le applicazioni nascono in parallelo agli smartphone. Al contrario di come si può comunemente pensare, non è l'iPhone il primo smartphone della storia, ma risulta molto complicato definire chi sia stato l'inventore del primo "telefono intelligente".
E' molto sottile la linea che divide il passaggio dai cellulari, muniti di poche funzionalità di comunicazione di base, agli smartphone, i quali hanno esteso le funzionalità dei primi telefoni. Già dalla metà degli anni '80, si parlava di "assistenti digitali personali", detti PDA, dispositivi portatili in grado di permettere l'esecuzione di altre funzionalità oltre alle chiamate, in cui molte aziende iniziarono a vedere del potenziale.
Intorno al 1993 Apple sviluppò il *Newton*, PDA connettibile alla rete Wifi, IBM rilasciò il suo *Simon*, la Palm i *PalmPilot* e un kit software grazie al quali chiunque avrebbe potuto sviluppare nuove app per i loro PDA, la Siemens il *GS 88 Penelope* grazie al quale nacque il termine "smartphone". [3] Nel 1997 uscì il Nokia 6110 portando su uno smartphone il gioco *Snake*, che molti considerano la prima vera applicazione per come le conosciamo oggi.

Non si può quindi definire con esattezza la data della nascita delle applicazioni. Si può però indicare con chiarezza che l'anno che ha portato all'esplosione del mercato delle app è stato il 2008, che coincide con l'apertura dell'Apple Store da parte di Apple, inserito nella versione 2.0 del sistema operativo per iPhone iOS. In un solo anno sono stati effettuati 1,5 miliardi di download e il numero ha continuato a crescere per ogni anno successivo [4].
Un altra protagonista responsabile del boom di questo mercato è stata Google, che nel 2008 ha lanciato il suo primo smartphone, con sistema operativo Android, dotato del relativo negozio online di app: l'Android Market. Esso ha preso successivamente il nome di "Google Play Store". Il *market* di Google ha impiegato 2 anni per raggiungere il miliardo di download, ma dal 2013 ha uguagliato l'Apple Store a 50 miliardi, e da quell'anno mantiene la leadership nella classifica. [5]

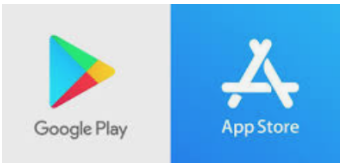

Figura 2.1: Loghi di Google Play e App Store

Seguendo il modello del cambiamento tecnologico di Brian Winston, professore e giornalista esperto di evoluzione digitale, potremmo definire le applicazioni come tecnologie *spin-off* degli smartphone. La nascita delle app infatti è paragonabile a quella del software: senza gli smartphone non sarebbero nate le app, così come senza il persona computer, non si sarebbero diffusi i software. I due mondi però oggi vivono strettamente in simbiosi. Le app hanno bisogno degli smarpthone per funzionare, ma quest'ultimi non troverebbero un'applicazione concreta senza le app.
E' vero infatti che al giorno d'oggi le aziende che producono smartphone devono il loro successo agli sviluppatori di terze parti, i quali grazie alle loro app, contribuiscono ad aumentare il numero di utenti ed espandere il mercato, sia delle applicazioni, sia degli smartphone.
Come ci insegnano gli studi delle rivoluzioni tecnologiche, una invenzione non si diffonderà con successo se non vi è una reale necessità sociale. Così come sono serviti 50 anni dall'invenzione del primo computer interamente digitale, *L'Atanssof-Berry Computer* (ABC) del 1939 [6], alla diffusione globale del personal Computer, anche per assistere all'espansione del mercato delle applicazioni, abbiamo dovuto aspettare diversi anni, fin quando le aziende più all'avanguardia sono riuscite a soddisfare un bisogno sociale sempre più crescente, come quello di svolgere qualsiasi tipo di funzione alla portata del proprio smartphone.

2.2 Mercato e numeri

Per capire l'impatto che le applicazioni hanno avuto sulla società nell'ultimo decennio possiamo farci aiutare dai numeri. Il dato che colpisce maggiormente è quello della crescita del mercato delle app : nel 2008 non esisteva, nel 2016 valeva 1300 miliardi di dollari, nel 2021 potrebbe diventare la terza economia a livello mondiale, con una stima di 6350 miliardi di dollari [7].

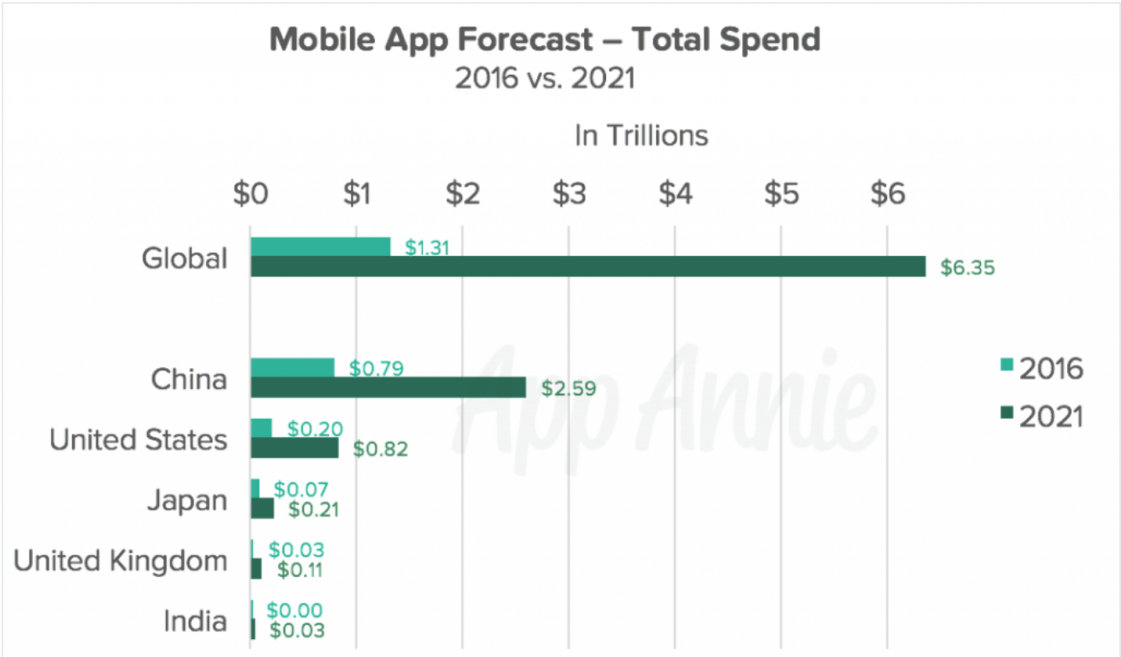

Figura 2.2: Valore del mercato intorno alle app nel mondo, confronto tra 2016 e previsione 2021. Fonte AppAnnie.com

Nel 2019, secondo la società App Annie, specializzata nella raccorda dati relativi al mondo delle app, si sono contati 204 miliardi di download a livello globale in tutti gli store, e si calcola che ogni utente passi in media 3,7 ore al giorno sul suo smartphone, il quale è diventato il canale principale di comunicazione. Dal 2016 infatti, gli accessi ad internet da smartphone e tablet hanno superato quelli da pc, e i nuovi *device* stanno contribuendo a portare anche la fetta di popolazione esclusa da internet, a navigare sul web, vista la loro accessibilità.

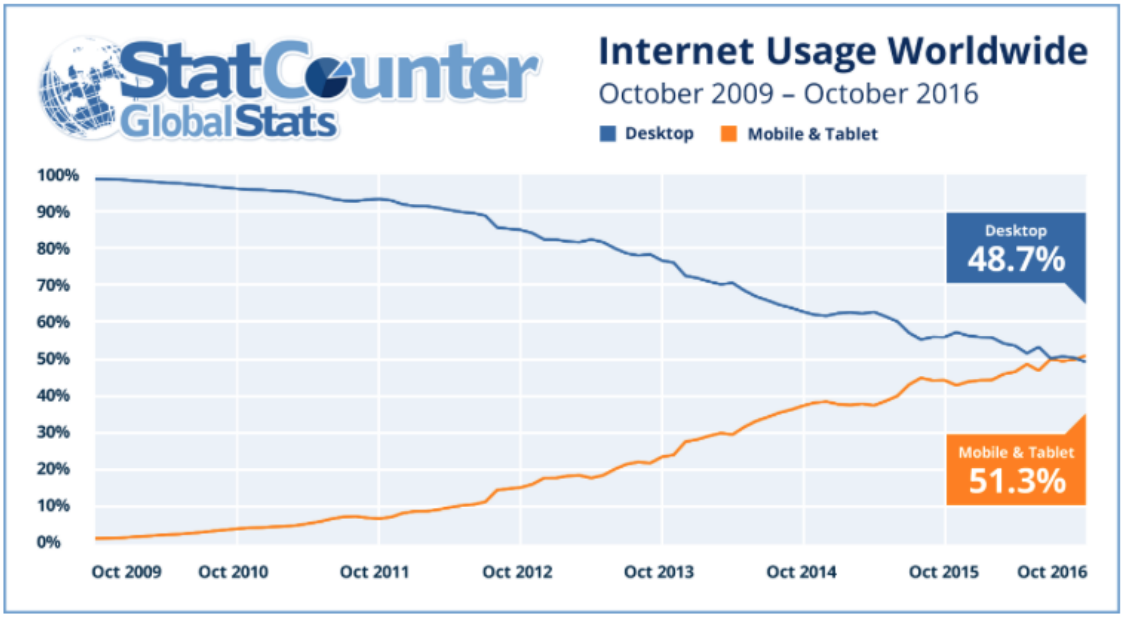

Figura 2.3: Confronto dell'utilizzo di internet da desktop o device mobili, dal 2009 al 2016, anno del cambio di tendenza. Fonte: StatCounter

I leader del mercato sono i paesi orientali, spinti dall'ingente produzione di smartphone presente nei loro territori. Prima fra tutti la Cina, per cui si prevedono 120 miliardi di download entro il 2022, seguita da India, Stati Uniti e Brasile, mentre sono Cina e Stati Uniti le nazioni in cui si spende di più per acquisti sugli store [7].
Questi numeri possono fare impressione ma serve specificare il contesto e il trend per capire come si è arrivati ad essi. Uno strumento utile a questa analisi è l'*App Maturity model*.
Osservando il grafico in figura 2.4 possiamo notare lo sviluppo del mercato legato alle applicazioni in funzione degli interessi degli utilizzatori, definendo 3 passaggi fondamentali :

- **Sperimentazione** : inizialmente gli utenti erano interessati a nuove scoperte e si era alla ricerca di nuove soluzioni applicative. A questo livello la principale fonte di guadagno è stato il costo del singolo download.

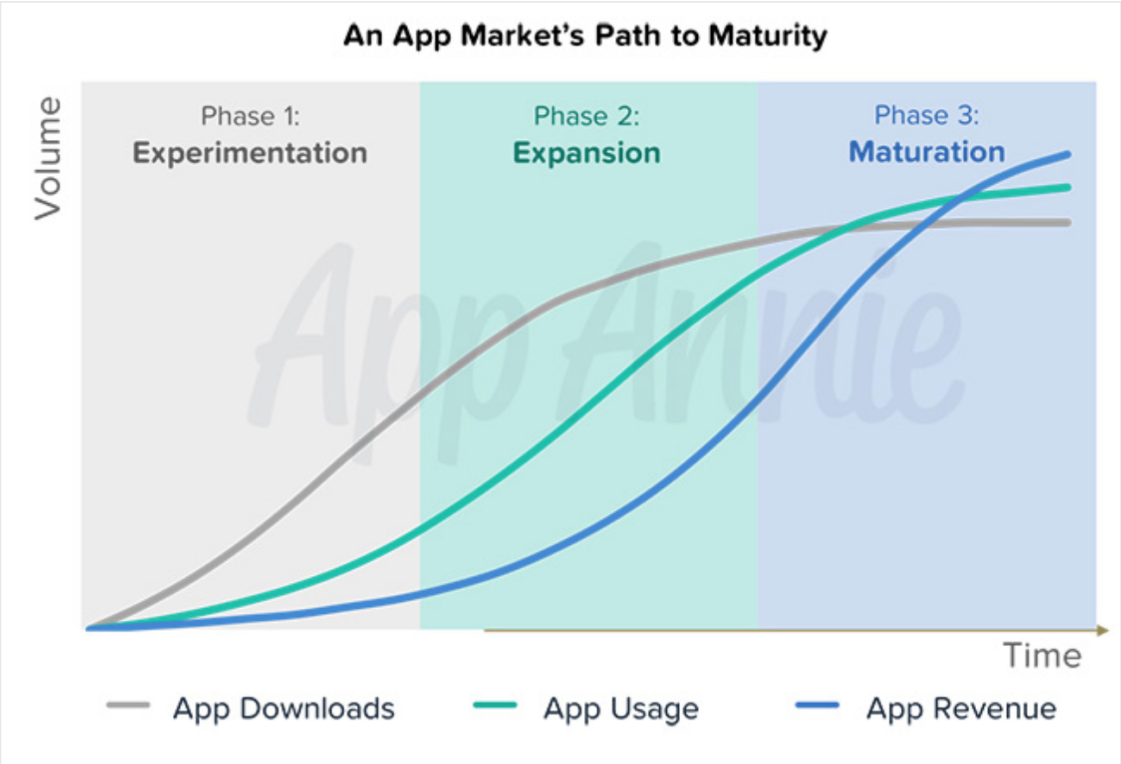

Figura 2.4: Modello di maturazione delle app secondo la società AppAnnie

- **Espansione** : In questo periodo i consumatori hanno capito quali erano i veri bisogni che un app poteva soddisfare e si sono concentrati su alcuni servizi principali (social, giochi, prenotazioni). E' sceso il numero di ricerche verso nuove app.

- **Maturazione** : E' sceso totalmente l'interesse per le novità, e gli utenti preferiscono spendere il tempo nelle loro app preferite, nelle quali spendono sempre di più. Migliora il modello di monetizzazione e si espandono servizi come pubblicità e mobile-commerce.

Oggi infatti solo una piccola percentuale della *app-economy* riguarda strettamente il prezzo delle applicazioni; Il grosso degli introiti viene generato dagli acquisti-in-app, pubblicità e servizi vari. Questo si può notare dal fatto che le app più utilizzate al giorno d'oggi sono gratis, come WhatsApp, Facebook, Snapchat o Youtube.

Come per quanto accade per la navigazione Web, la maggior parte degli introiti finisce in mano ad un gruppo ristretto: l'1,6% delle aziende guadagna più del restante 98.4% degli sviluppatori, mentre il 97% del tempo è trascorso sulle 10 app più utilizzate nel paese di riferimento [8].
I Due principali negozi di applicazioni sono accessibili attraverso i sistemi operativi più diffusi: App Store per iOS e Google Play per Android.
Su un totale di 6,5 milioni di app, Google Play ne possiede 2,8 milioni, mentre App Store 2,2 milioni. La restante percentuale è spartita fra competitors minori, come Windows Store, Amazon Appstore e BlackBerry World [9].
Il negozio di Apple però supera più del doppio Google per quanto riguarda i ricavi, totalizzando 55.26 miliardi di dollari nel 2019, dove per entrambe le piattaforme è la categoria dei giochi ad essere la fonte più redditizia, per acquisti-in-app e prezzo di download. [9].
L'Europa segue le grandi potenze mondiali, che stanno davanti anche grazie alla popolazione più elevata. Il principale attore europeo in questa scena è il Regno Unito, che si colloca al terzo posto della classifica globale per il maggior numero di sviluppatori e il 6,8 del valore complessivo del mercato.
Per quanto riguarda la situazione italiana i numeri non possono competere con le grandi potenze mondiali, ma sono significativi i dati relativi all'utilizzo delle app e dei loro servizi che stanno guadagnando velocemente terreno rispetto alle altre tecnologie, per quanto riguarda l'informazione e l'intrattenimento, seguendo l'andamento globale.
Ogni mese in Italia oltre 32 milioni di Italiani utilizzano smartphone o tablet per collegarsi ad internet, che equivale all' 82% della popolazione italiana che sfrutta i servizi *on-line*. La pubblicità su dispositivi mobili occupa circa la metà (47%) della promozione on-line, che equivale a 1,4 miliardi di euro, e il 17% del totale giro di affari comprendente di tutti i canali (tv, radio, stampa, Internet) [10].
Un altro dato importante che proietta il mondo delle applicazioni verso una crescita futura è quello legato alla Generazione Z, i nati nella seconda metà degli anni '90 e le fine degli anni 2000. Questa fetta di popolazione spende più ore sulle app che quella più anziana. Gli utenti con una media di età fra 18 e 24 anni, utilizza le applicazioni per più di tre ore al giorno, mentre le persone con età superiore a 55 anni, trascorrono meno di due ore [8].

Le applicazioni sono quindi uno strumento molto importante al giorno d'oggi, sempre più presenti all'interno delle nostre vite lavorative e non, e come dimostrano i numeri, il numero di utenti e il giro di affari è destinato a crescere. Per questo esse si stanno facendo strada tra i canali di comunicazione più utilizzati : nel 2019 lo smarptohne per gli utenti Americani ha superato il tempo di utilizzo della TV [11].

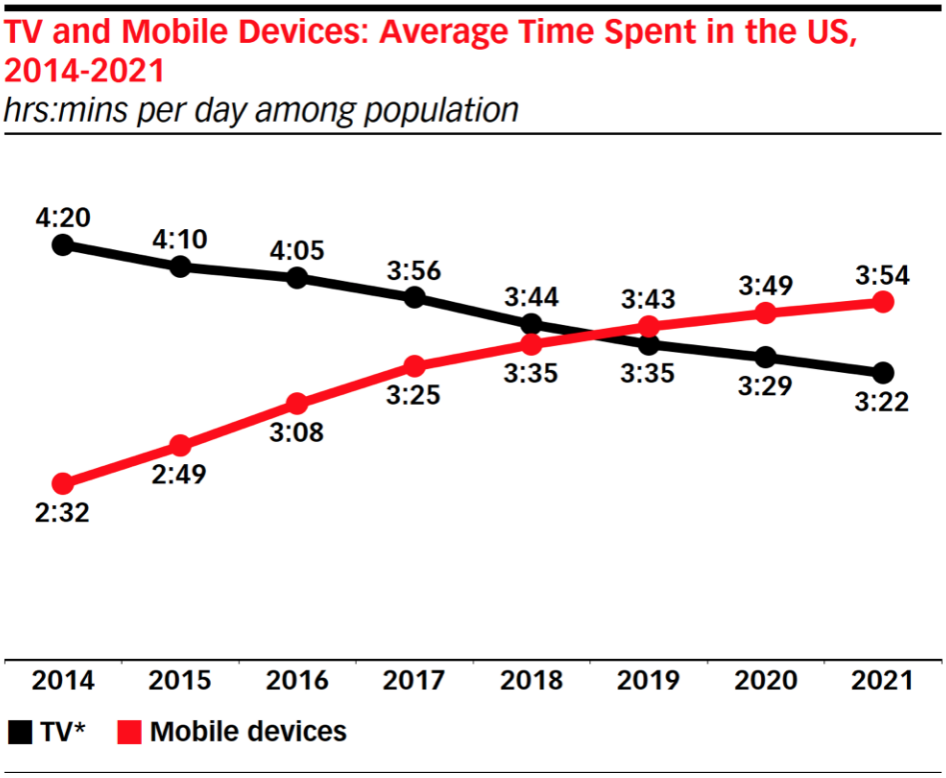

Figura 2.5: Media ore giornaliere trascorse utilizzando la TV rispetto ai dispositivi mobili in USA. Fonte: eMarketer.com

Le aziende quindi dovranno adattarsi a questo cambiamento se vorranno stare al passo con i tempi, portando la loro presenza all'interno degli smartphone per mezzo delle applicazioni. Oggi quinid non è più sufficiente essere presenti sul web, risultando visibile ai motori di ricerca, ma diventa necessario presentarsi al cliente anche sottoforma di applicazione, per il maggior numero di device possibili.

2.3 Sistemi operativi e piattaforme mobili

Visto il largo utilizzo delle applicazioni, molte aziende sentono la necessità di entrare in questo mercato, sviluppando nuovi software applicativi o adattando quelli già presenti per i dispositivi mobili. Il problema quindi si sposta alla tecnologia da utilizzare per sviluppare la propria applicazione. Come succede da sempre con le tecnologie informatiche, non vi è mai un solo approccio al problema, ma la soluzione migliore è data da un'insieme di fattori che possono sbilanciare la scelta in una direzione piuttosto che un'altra.
Quando si intende sviluppare una applicazione bisogna decidere a priori quale clientela si vuole soddisfare, oppure fare una scelta universale.
Le principali differenze stanno nella scelta del sistema operativo su cui dovrà essere eseguita l'app e di conseguenza la tecnologia utilizzata per svilupparla.
Nei paragrafi seguenti verranno elencate le differenze principali fra i due sistemi operativi mobile che oggi occupano l'intero mercato: iOS e Android. E' importate specificare che anche Windows Phone ha avuto un ruolo importante, seppur con molti meno utenti, all'interno dei sistemi operativi per smartphone e tablet. Microsoft ha però annunciato che dal 10 dicembre 2019 non avrebbe più supportato con aggiornamenti e assistenza i suoi utenti, consigliando di passare ad Android e iOS.
Per questo, all'interno di questa tesi, ogni volta che si farà riferimento alle piattaforme o sistemi operativi per smartphone, si intenderanno solamente iOS o Android, tralasciando gli aspetti ormai in via di estinzione che riguardano Windows Phone o altri sistemi minori.

2.3.1 Caratteristiche Android

Per sviluppare applicazioni in Android i programmatori possono scegliere se utilizzare i linguaggi *Java*, *C++* o *Kotlin*. I principali framework di sviluppo utilizzati sono Android SDK, kit di sviluppo connesso ad *Android Studio*, e *Android JetPack*, un framework con componenti Android pre compilati.
In seguito sono riportati alcuni dei vantaggi e svantaggi dello sviluppo di applicazioni in Android rispetto ad altre piattaforme .

Vantaggi

- **Open Source** : il sistema Android è aperto, il codice sorgente è disponibile e personalizzabile dalle case produttrici di smartphone. Inoltre gli sviluppatori Android hanno l'accesso a più funzioni, che talvolta sono limitate per le app iOS.

- **Design** : gli sviluppatori possono seguire molte linee guida messa a disposizione da Google attraverso i suoi framework, come Material Design, per creare facilmente una *user interface* (UI) e *user experience* (UX) adatta all'applicazione. Il design dell'app resta comunque molto flessibile e a discrezione dello sviluppatore, il quale può scegliere anche di non seguire i pattern pre impostati.

- **Numero dispositivi** : la politica open source di Google per Android ha portato a conquistare tutte le case produttrici di dispositivi, mobili e non. Oggi Andoird è presente su smartphone e tablet, ma anche nelle TV, nelle automobili e negli orologi.
  Anche Apple si è mossa in questa direzione, ma il vantaggio di Android è la sua versatilità : iOS può essere installato solo su dispositivi Apple, Android invece è scelto da tutte le altre case produttrici. Questa frammentazione può risultare un vantaggio in fase di programmazione, perché si possono sviluppare app per una gamma più vasta di dispositivi.

- **Pubblicazione** : Le app Android sono più facili da pubblicare sul Google Play store, perché sono soggette a meno vincoli e controlli. E' possibile pubblicare sullo store anche in poche ore.

Svantaggi

- **Frammentazione** : gli svantaggi di sviluppare in Android sono legati alla vasta diversità di dispositivi che lo utilizzano. Se la frammentazione può essere un vantaggio per raggiungere un grosso numero di utenti, può diventare uno svantaggio quando bisogna adattare il codice, soprattutto il *front-end* e gli strumenti hardware, al device in cui Android è installato.

- **Test** : il temo impiegato per testare le applicazioni può aumentare, per permettere agli sviluppatori di analizzare il comportamento dell'app su dispositivi e versioni differenti.

- **Tempi e costi** : I tempi di sviluppo e test aumentano, facendo aumentare anche i costi.

2.3.2 Caratteristiche iOS

La politica di Apple è sempre stata opposta. Il suo sistema operativo è utilizzabile solo su dispositivi proprietari e non è personalizzabile in alcun modo. Questo sistema permette ad Apple ad avere un maggior controllo sui device, concentrandosi sul design e l'usabilità dei suoi strumenti, che da sempre è il punto di forza per la casa di Cupertino.
I linguaggi di programmazione utilizzati sono *Swift* o *Objective-C*. Per sviluppare app sono utilizzati alcuni framework. Uno fra tutti è *Xcode*, l'IDE ufficiale rilasciato da Apple, ma sono utili anche altri strumenti, come *Swift Playgrounds*, per lo sviluppo nell'omonimo linguaggio, o *TestFlight*, utile per effettuare test.

Vantaggi

- **Ricavi** : come analizzato nel capitolo 2.2 , gli utenti di applicazioni Apple spendono di più rispetto alle applicazioni Android, sia a livello di acquisti-in-app, che di costo della singola applicazione.

- **Numero di dispositivi** : uno sviluppatore saprà perfettamente le caratteristiche e le versioni di tutti i dispositivi che andranno ad ospitare la sua applicazione, perché sono solamente quelli prodotti da Apple. Questo comporta un vantaggio sui tempi e i costi di sviluppo e di test.

Svantaggi

- **Pubblicazione e flessibilità** : la pubblicazione dell'applicazione sull'Apple Store è più complicata rispetto a quella su Google Play. Oltre a pagare un canone annuale, è necessario che l'applicazione rispetti delle linee guida piuttosto restringenti in termini di funzionalità, contenuto e prestazioni, che in genere rallentano i tempi di pubblicazione.

2.3.3 Scelta della piattaforma

Quando si intende scegliere per quale piattaforma verrà pubblicata l'applicazione, ci sono altri aspetti da tenere in considerazione oltre a quelli tecnici, discussi nel paragrafo precedente.
Un'aspetto fondamentale è il target a cui l'applicazione è rivolta. Per definire il target, tornano utili i dati demografici sull'utilizzo delle App. Chi scarica da Apple Store è tendenzialmente più giovane e disposto a spendere di più, perché vive in zone del mondo più ricche che offrono migliori salari. Android invece detiene il primato nei paesi in via di sviluppo e nelle zone di basso reddito [12].
Un'altro aspetto importante è il costo assoluto per lo sviluppo di un'app per singola piattaforma. Mediamente i prezzi per lo sviluppo di applicazioni native Android sono superiori, dovuti al tempo di adattamento a tutte le versioni disponibili. Il numero limitato di dispositivi iOS, invece, accelera i tempi di sviluppo.

Si possono riassumere gli aspetti trattati in questo paragrafo delineando i fattori principali che potrebbero indirizzare la scelta verso una piattaforma piuttosto che un'altra.
In genere la scelta dovrebbe ricadere su iOS se si desidera generare maggiori entrate per per ciascun utente, per seguire un processo di sviluppo dell'applicazione più semplice e standardizzato ed avere maggiori garanzie sulla privacy e sicurezza dei dati degli utenti, vista la natura closed source e le linee guida molto restrittive di Apple.
Android invece risulta la scelta migliore nel caso si voglia raggiungere un pubblico più ampio, maggiore flessibilità nella personalizzazione dell'app e rapidità di pubblicazione nello store.
La scelta della piattaforma a cui sarà rivolta l'applicazione è importante e va fatta a priori, in funzione degli obiettivi del servizio offerto dall'app. In questo capitolo si sono discusse le caratteristiche e i punti di forza e debolezza dei due sistemi. La scelta di un sistema porta sempre dei compromessi, ma nulla vieta alle aziende e ai programmatori indipendenti, di non scegliere, e sviluppare le proprie applicazioni per entrambi i sistemi operativi.

2.4 Sviluppo

Una volta che si è a conoscenza degli aspetti legati ai sistemi operativi, è importante analizzare gli strumenti a disposizione che permettono lo sviluppo concreto delle applicazioni.

1. approccio nativo
2. approccio multipiattaforma

1. Le applicazioni native sono le app sviluppate esclusivamente per una piattaforma, utilizzando il linguaggio e gli strumenti di sviluppo appartenenti al sistema di riferimento. Sono pubblicate nei relativi store, e qualora si desideri raggiungere sia dispositivi Android che dispositivi Apple, è necessario creare e mantenere due progetti totalmente diversi, aumentando così i tempi di sviluppo i costi e le risorse necessarie alla gestione.
Le applicazioni native però godono di migliore reattività e affidabilità, permettendo di offrire la migliore esperienza raggiungibile all'utente. Esse sono in grado di supportare il funzionamento offline e offrono prestazioni migliori e semplicità nell'accesso alle funzioni del telefono o tablet, come fotocamera, accellerometro ecc..

2. Con il termine multipiattaforma invece si indicano le applicazioni che, con la stessa base di codice, possono essere eseguite sia su iOS che su Android. Rispetto a quelle native sono più facili e rapide da sviluppare e generalmente costano di meno. La manutenzione dell'applicazione è più semplice perché esiste una sola versione da aggiornare. Le prestazioni e l'user experience però non hanno ancora raggiunto il livello delle applicazioni native, anche se i nuovi framework e la potenza dei nuovi device, permettono di creare soluzioni valide.
Le applicazioni multipiattaforma a loro volta possono essere divise in altre sottocagorie, a seconda della tecnologia utilizzata per fornire il servizio *cross-platform*, che verranno analizzate nel capitolo 3.

Capitolo 3

Multipiattaforma

3.1 Perché scegliere il multipiattaforma

La volontà delle aziende di raggiungere il maggior numero di clienti, ha portato una crescita considerevole dell'approccio multipiattaforma allo sviluppo delle applicazioni, per poter marcare la propria presenza sia su Apple Store che su Google Play.
Il vantaggio che questo approccio comporta in termini di gestione, tempi e costi sono considerevoli e superano molte volte gli svantaggi, relativi alle prestazioni e all'interfaccia con i vari device.
L'approccio multipiattaforma presenta alcuni benefici che dovrebbero indirizzare la scelta della tecnologia di sviluppo, se consideriamo la necessità di sviluppare una applicazione per entrambe le piattaforme, iOS e Android.

- **Portabilità del codice**: è sicuramente l'aspetto più importante. Il codice è scritto una sola volta ed è portabile su diverse piattaforme. Garantisce la metà dello sforzo rispetto all'approccio nativo.

- **Costi**: minor costo di sviluppo, si scrive un solo codice e non servono due team di programmatori specializzati.

- **Consistenza della UI**: i framework multipiattaforma adattano automaticamente i componenti grafici al sistema nativo di riferimento

- **Cloud**: l'integrazione con l'ambiente cloud è semplice e supportata dai framework.

- **Linguaggi**: i programmatori non devono affrontare ostacoli che li obbligano a imparare linguaggi specifici, come *Objective-C* o *Swift* ma sono necessarie competenze molto più diffuse come HTML e *Javascript*.
- **Tempistiche**: dovendo sviluppare un solo prodotto, si può entrare con tempi brevi sul mercato, sfruttando i trend del momento, che in questo settore possono cambiare velocemente.

Il problema dello sviluppo multipiattaforma invece risiede principalmente nelle prestazioni. Qualsiasi approccio, come quello Web o ibrido, rallentano, l'esecuzione delle applicazioni, perché l'interfaccia con le API native non avviene direttamente ma attraverso strumenti intermedi che permettono allo strato di codice comune di alto livello di interfacciarsi con i servizi del device. Questo passaggio rallenta invevitabilmente l'esecuzione delle applicazioni, che ne risentono in base alle funzioni che compiono. Per applicazioni leggere, che lavorano principalmente sui dati, la differenza sarà impercettibile, mentre altre app più complesse potrebbero fornire esperienze scadenti se sviluppate con tecnologie multi-piattaforma.

3.2 Tecnologie

3.2.1 Approccio Web e PWA

Lo sviluppo web è cambiato molto attraverso gli anni. La prima rivoluzione l'ha portata AJAX, comparso nei primi anni 2000, trasformando il web, da un insieme pagine statiche, a una esperienza dinamica. Gli utenti si sono abituati a non dover più aspettare decine di secondi per caricare una pagina, ma di interagire con essa in tempo reale, come ad esempio effettuare lo zoom e muoversi trascinandp il mouse, all'interno di una mappa di Google.
Sfortunatamente, i progressi raggiunti dal Web in versione desktop non si sono tradotti efficacemente nel mondo mobile. Infatti gli utenti di smartphone e tablet utilizzano i servizi sotto forma di applicazioni scaricate dagli store, evitando il più possibile i browser, anche se molte volte il servizio è disponibile in entrambe le piattaforme.

Una ricerca di *ComScore* riporta infatti che gli utenti trascorrono il 10% del loro tempo online sui browser, mentre il 90% all'interno delle app, per la migliore user experience, le notifiche push, e la facilità di trovarle sulla home.
Nonostante questi fattori, le soluzioni web stanno conquistando terreno nel mondo dello sviluppo *mobile* negli ultimi anni.
L'utilizzo delle applicazioni è estremamente concentrato : gli utenti spendono il 50% del loro tempo in una specifica app, il 78% nelle 3 app personalmente più utilizzate, e il 97% nelle 10. Inoltre più del 50% degli utilizzatori di smartphone e tablet negli Stati Uniti ha dichiarato di non scaricare alcuna nuova app al mese [13].
Gli utenti, alla luce di questi numeri, installano e usano poche app e oggi è difficile ritagliarsi un posto fra esse, visti i colossi coi quali bisogna competere. In contrasto a questi dati però, i siti web visitati da *mobile* sono oltre cento per ogni mese, per la potenza e la facilità offerta dai link, che spostano la navigazione da un sito all'altro, all'interno del browser. Su questo dato fanno leva le applicazioni mobile sviluppate con approccio Web.
Tecnicamente, le applicazioni Web funzionano senza essere installate nel dispositivo ma sono eseguite nel browser, proprio come un sito web. Presentano tutte le caratteristiche di un sito, come l'url, i link, il modello client-server, i linguaggi Javascript, CSS e HTML e la loro interfaccia è adattata alle dimensioni del dispositivo su cui viene eseguito il browser.
Il peso computazionale e la memoria utilizzata sul dispositivo saranno estremamente basse, in quanto il grosso del lavoro è svolto dai server di *back-end*. Questo tipo di applicazioni non saranno scaricabili dagli store, ma come riportato dai dati precedenti, questo potrebbe non essere più un problema, visto il poco interesse del pubblico a nuovi download.
L'approccio nativo ha offerto negli anni la soluzione migliore per lo sviluppo app, per la migliore esperienza d'utilizzo che offrono. Quando vengono aperte, esse si caricano più velocemente, di solito lavorano anche offline, possono mostrare notifiche, sincronizzarsi in background e avere accesso ai sensori, come fotocamera e accelerometro. Ma la portata è piuttosto limitata, richiedendo una versione diversa per ogni piattaforma. Il web dal canto suo è più protetto perché il suo modello è più rispettoso della privacy, rispetto ai protocolli di comunicazione proprietari delle app native, ma finché il web non è riuscito a offrire l'accesso alle funzionalità delle

app native, la sua raggiungibilltà non è stata sufficiente per imporsi come tecnologia principale.
L'obiettivo delle web app oggi è quello di coniugare le abilità e l'esperienza a cui gli utenti sono abituati sulle app native, nel web, sfruttando i punti di forza di entrambi.
Per questo, dal 2017, è stato introdotto il concetto di PWA, le *Progressive Web App*, che stanno iniziando a diffondersi. Esse puntano a raggiungere la qualità del servizio offerto dalle app native, mantendo un approccio web.
Possiamo definire le PWA come semplici applicazioni web specializzate all'utilizzo *mobile*, concentrate a migliorare alcuni fattori:

- **Velocità**: migliorano il caricamento e la fluidità delle app web.
- **User Experience**: permettono di installare un'icona sulla home e accedere all'applicazione senza utilizzare il browser. Forniscono un'interfaccia a schermo intero eliminando le funzionalità di contorno dei browser.
- **Integrazione**: offrono l'accesso all'hardware a livello del nativo. Con le PWA è possibile interagire con i sensori, inviare notifiche *push*, utilizzare API per pagamento, e interfacciarsi con le altre app.
- **Affidabilità**: è gestita la sincronizzazione delle app anche offline, permettendo all'utente di utilizzare il servizio anche in caso di connessioni lente o assenti, evitando le spiacevoli attese delle pagine web. Le PWA si servono dei *Service Worker*, degli script che lavorano in background alle sessioni web e si occupano della sincronizzazione dei dati in caso di connessioni lente o assenti, fornendo servizi di caching per l'accesso ai dati.

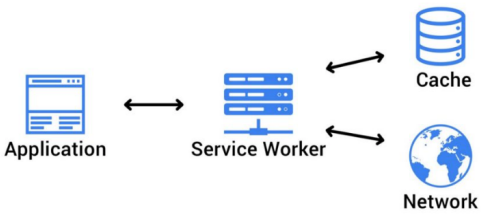

Figura 3.1: Schema logico di funzionamento dei Service Worker

Le PWA sono un'ottima soluzione al giorno d'oggi e stanno aumentando il numero di aziende che scelgono di sviluppare la propria, con ottimi risultati, come Twitter, Pinterest, Trivago, Zalando.

3.2.2 Approccio ibrido

Un'altra solzione per creare un'applicazione multipiattaforma è data dall'utilizzo dell'approccio ibrido.
Il termine ibrido sta ad indicare una via di mezzo, tra il mondo nativo e quello delle web app. L'idea che sta alla base è quella di utilizzare dei componenti nativi in grado di visualizzare pagine web, che non sono altro delle web app con funzioni base, che si interfacciano alle funzionalità del device tramite il contenitore nativo. Lo sviluppatore deve quindi creare l'applicazione seguendo i processi di una qualsiasi web app, con i linguaggi HTML, e Javascript.
Per visualizzare i contenuti le applicazioni ibride utilizzano le *WebView* (che possono cambiare nome a seconda dell' OS), che sono componenti nativi in grado di effettuare il rendering del codice e visualizzare la grafica all'interno di un applicazione.
L'interfaccia con il sistema sottostante invece è assegnata ai cosiddetti *Wrapper*, diversi per ogni sistema operativo, che interpretano il linguaggio Javascript e l'interfaccia con l'hardware. I *Wrapper* possono essere scritti dal programmatore stesso, oppure ci si può servire di framework già esistenti che forniscono le API Javascript necessarie all'applicazione per poter accedere alle funzionalità di basso livello.
Esistono diversi framework che si occupano di gestire l'interfaccia con il dispostivo, evitando questi problemi al programmatore, che può concentrarsi allo sviluppo del codice comune dell'applicazione. I framework sono più o meno specializzati a gestire i vari serivizi dei dispositivi e se necessario, possono essere utilizzati insieme, per estendere le funzionalità.
Le applicazioni ibride presentano quindi una valida scelta per lo sviluppo multipiattaforma, tenendo conto che i principiali framework, come *Ionic*, *Cordova* e *PhoneGap*, hanno ormai una dettagliata documentazione e una community molto attiva, essendo sul mercato da qualche anno.

Cordova

Cordova è il framework che gestisce app ibride più diffuso. Esso merita una breve analisi in quanto le sue funzionalità sono implementate dal framework che ho utilizzato per lo sviluppo del progetto relativo a questa tesi.
Cordova è un framework gratuito e open-source, rilasciato da *Apache*, che permette di sviluppare applcazioni cross-platform, sfruttando le tecnologie e gli standard web (HTML5, CSS3 e Javascript), offrendo l'interfaccia Javascript per connettersi alle API native del device e supportando il funzionamento offline delle applicazioni.

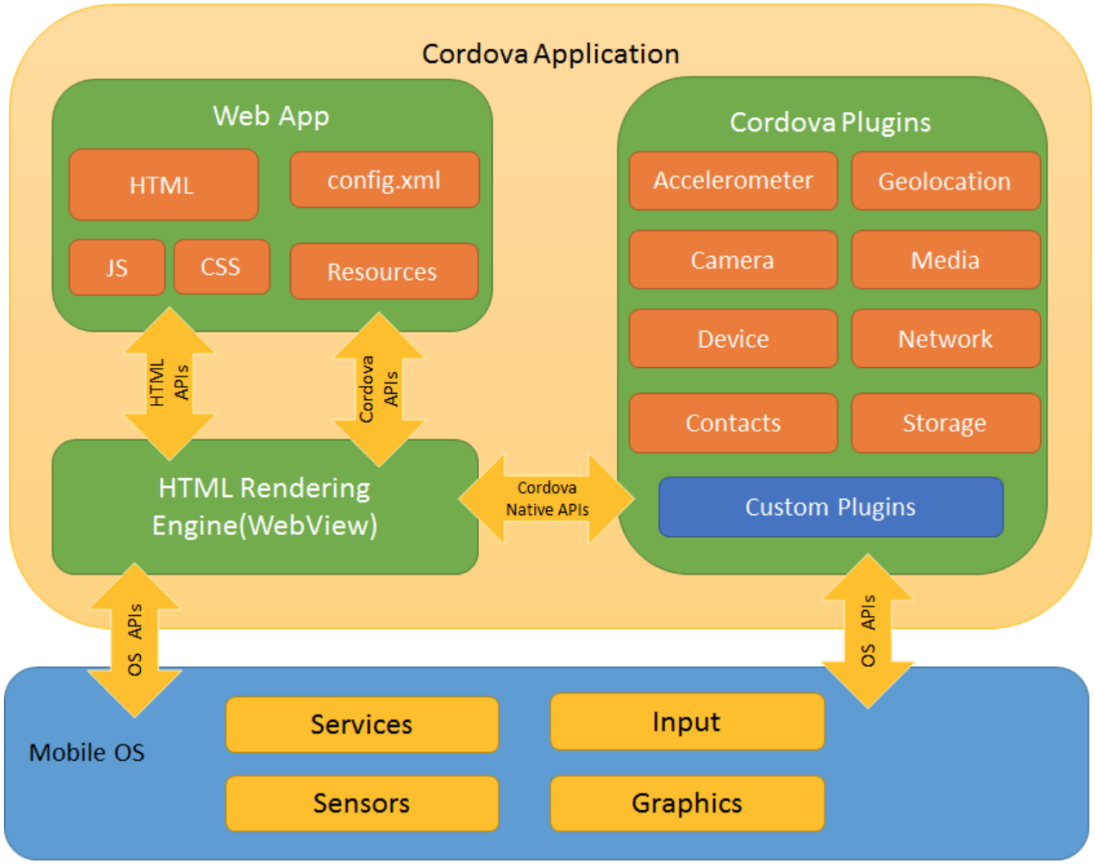

Figura 3.2: Architettura di Apache Cordova. Fonte: apache.cordova.org

L'immagine 3.2 riporta l'architettura del framework, che segue la logica descritta per le applicazioni ibride.
Nel blocco relativo alla *Web App*, risiede il codice dell'applicazione. Essa, come discusso in precedenza, è implementata a tutti gli effetti come una pagina web, necessitando quindi di un file locale, chiamato index.html, che fa riferimento al CSS e Javascript, immagini e altri file necessari per l'esecuzione.
L'applicazione è eseguita quindi all'interno di una *WebView*, la quale effettua il rendering del codice HTML fornendo l'interfaccia utente. La *WebView* è collocata all'interno di un *wrapper* nativo, elemento che viene poi distribuito sui vari negozi di applicazioni.
Un file cruciale per la configuarazione dell'app all'interno del *wrapper* nativo è *config.xml*, che fonrnisce le principali informazioni legate all'applicazione, impostandone i parametri che ne determineranno il funzionamento, come ad esempio l'abilitazione della variazione di orientamento, o le versioni dei sistemi operativi supportati.
Strumenti molto utili in fase di programmazione sono i *plug-in*. Essi, una volta installati, forniscono un'interfaccia di comunicazione fra Cordova e i componenti nativi, collegando anche le API standard dei dispositivi. In questo modo si possono richiamare le funzioni native direttamente utilizzando il codice Javascript, utilizzando semplici metodi appartenenti a classi che rappresentano i servizi del device, come sensori, dati e rete.

```
1 navigator.camera.getPicture(cameraSuccess,
    cameraError, cameraOptions)
```

Listato 3.1: Esempio in javascript dell'utilizzo del plugin Cordova per la cattura di una foto attraverso la fotocamera. La funzione accetta come parametro le opzioni e le callback da chiamare in caso di successo ed errore

I *plug-in* sono automaticamente collegati alle API native del sistema operativo evitando così allo sviluppatore di risolvere i problemi di interfaccia di basso livello. Oltre ai plug-in *core* di Cordova, esistono diversi plug-in di terze parti che forniscono funzioni aggiuntive, che molte volte sono implementati per permettere l'utilizzo di componenti specifici, a seconda della piattaforma. I plug-in possono essere creati da ogni utente, per gestire qualsiasi componente nativo personalizzato.
Per quando riguarda il *front-end*, Cordova non fornisce alcun *Widget UI*

o framework per la composizione grafica, ma implementa solamente l'ambiente in cui essi possono essere eseguiti. E' possibile importare qualsiasi pacchetto esterno per la gestione dell' *user interface*, come *Ionic*, *Onsen* o *Material Design*.
Cordova è un framework molto utilizzato anche se molte volte agisce nelle retrovie. Molti framework, come PhoneGap, sfruttano le sue carattersitche, estendendone le funzionalità.

3.2.3 Altri framework

Negli anni si sono sviluppati molti framework, ognuno con il suo approccio al problema, ma tutti con l'obiettivo comune di permettere lo sviluppo di codice portabile in piattaforme diverse. I principali framework non ancora citati sono Xamarin e Flutter.

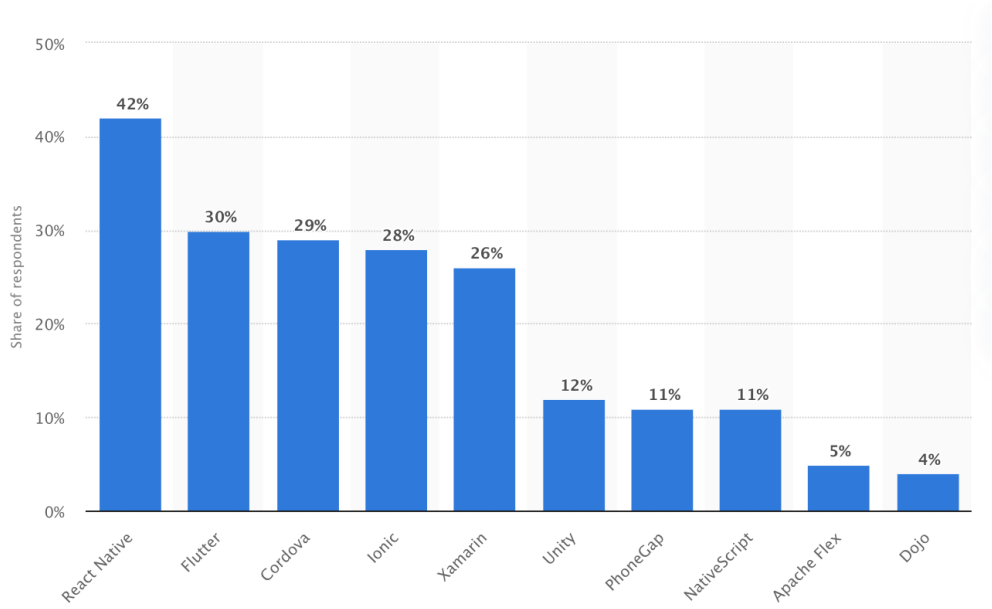

Figura 3.3: Classifica dei framework multipiattaforma per percentuale di utilizzo degli intervistati nel 2019. Fonte: Statista.com

Xamarin

Xamarin è un framework *open-source*, fondato nel 2011 e acquistato poi da Microsoft. Esso fornisce accesso alle API native, permettendo la portabilità del codice. Questa piattaforma ha un approccio cross-compilato; viene utilizzato per lo sviluppo il linguaggio C#, molto conosciuto perché adottato da Microsoft, che permette di ottenere buonissime prestazioni su iOS, Android e Windows, essendo disponibili le API native sottoforma di normali librerie C#.
Un *cross-compilatore* traduce in seguito i file sorgenti comuni, in file binari nativi.
Questo framework permette di raggiungere prestazioni simili a quelle native, e si integra alla perfezione con *Visual Studio* e .NET, piattaforme molto utilizzate dalle aziende, che possono sfruttarle, grazie a Xamarin, per creare applicazioni multaipiattaforma.
Un altro punto di forza è la community, composta da oltre 60.000 contributori da oltre 3,700 compagnie.
I principali difetti di Xamarin sono le dimensioni delle app, che raggiungono facilmente i 20MB anche per applicazioni semplici, per responsabilità del cross-compilatore che deve tradurre le chiamate C# in chiamate native.
Xamarin è nato a San Francisco, nel 2011, grazie a due ingegneri che hanno fondato l'omonima azienda. Negli anni, col diffondersi dell'interesse per lo sviluppo multipiattaforma, e al crescere della community di programmatori (oltre 1,2 milioni) e aziende, la società ha attirato l'attenzione delle grandi compagnie, e nel 2016 è stata acquisita da Microsoft. Da quell'anno1 data infatti Xamarin è incluso nella piattaforma open-source .NET, la quale è gratuita e non prevede costi di licenza, anche per uso commerciale.
Xamarin quindi rimane una buona opzione per applicazioni semplici, ma risulta pesante per progetti più complicati, come giochi o applicazioni che richiedono un utilizzo pesante dei sensori.

Flutter

Flutter è un framework orientato al *front-end* creato da Google, che permette di scrivere applicazioni per Android e iOS basate sullo stesso codice comune. Il linguaggio utilizzato per scrivere applicazioni in Flutter è il

Dart, anch'esso di proprietà di Google, nato per rimpiazzare Javascript nelle applicazioni web e poi indirizzato allo sviluppo *mobile*.
La struttura di Flutter permette al programmatore di svolgere molte azioni scrivendo codice di alto livello.
Come si può notare dall'immagin 3.4, l'architettura è divisa in 3 parti:

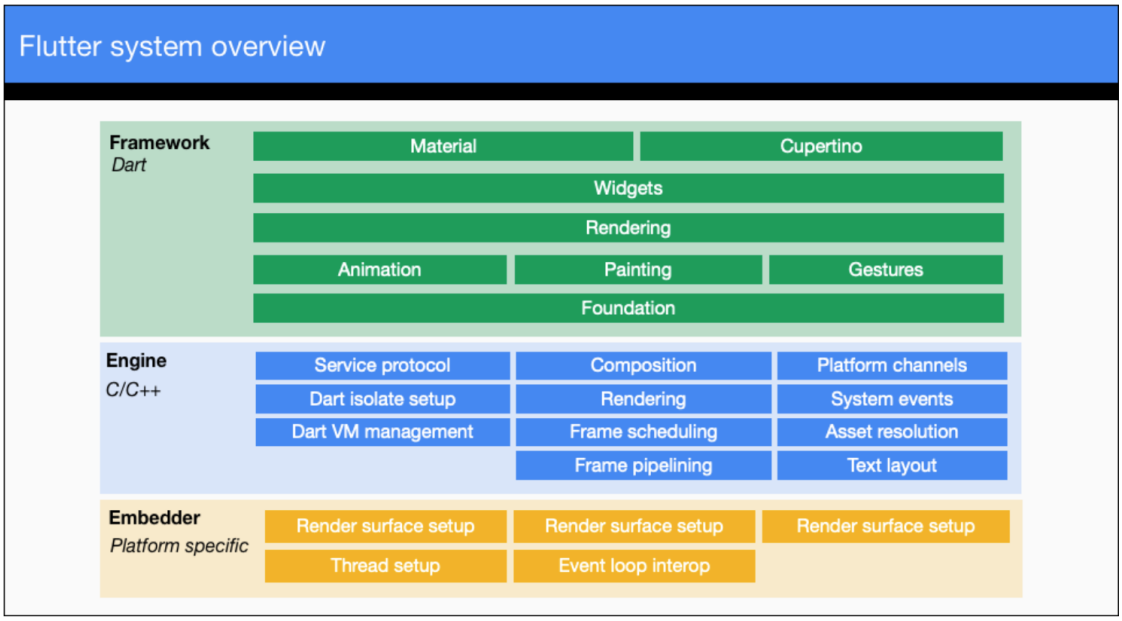

Figura 3.4: Architettura di Flutter. Fonte: flutter.dev

1. **Framework**: in questo livello viene scritta l'applicazioni in linguaggio Dart, servendosi delle librerie e strumenti astratti, senza preoccuparsi dell'interfaccia con il dispositivo.

2. **Engine**: scritto in C++, permette di effettuare il rendering dell'applicazione scritta in *Dart* a basso livello, utilizzando la libreria *Skia Graphics* di Google stesso. Fornisce la possibilità di effettuare degli "hot reload", che permettono di visualizzare in tempo reale le modifiche del codice all'interno dell'applicazione.

3. **Embedder**: livello più profondo dell'architettura, e interviene a livello nativo sulle funzioni del device della piattaforma specifica.

Flutter è quindi un ottimo strumento, che offre la possibilità di creare User Experience complete e avanzate, a livello delle native, con portabilità del

codice. Esso offre *widget*, API, e molti altri strumenti necessari per lo sviluppo di applicazioni per smartphone e tablet, permettendo sempre la personalizzazione anche dei componenti predefiniti.
Flutter riscuote molto successo perché è sotto il controllo di Google, ed è usato da Google stessa per sviluppare le sue applicazioni.
Inoltre è completamente gratuito ed open-source, ed è possibile installarlo sui principali sistemi operativi, quali Windows, macOS e Linux.
Il problema principale di Flutter è che non dispone di molte API native da usare, ed è necessario servirsi di pacchetti di terze parti.

3.3 Il Cloud

Il Cloud è uno strumento che ha influito molto sullo sviluppo multipiattaforma delle applicazioni.
Grazie al fatto che molti approcci sono basati su tecnologie web, risulta semplice integrare gli ambienti di sviluppo all'interno di soluzioni Cloud, fornendo agli sviluppatori un ambiente di sviluppo completo, dove oltre alla programmazione, può essere gestita l'infrastruttura per l'esecuzione, l'analisi e la distribuzione dell'applicazione.
Un esempio concreto è offerto da Google, che fornisce questo servizio grazie a *App Engine* e *Firebase*, che combinati insieme offrono la possibilità di programmare un app e distribuirla, sfruttando i servizi Cloud.

3.3.1 Cloud Computing

Con *Cloud Computing* si indica l'erogazione di un servizio offerto *on demand* da un fornitore ad un cliente finale attraverso la rete, come l'archiviazione, l'elaborazione o la trasmissione dei dati [14].
Oggi si può quindi affidare ad un azienda esterna specializzata, la gestione di risorse informatiche che vengono messe a disposizione via web, senza doversi accollare il problema dell' acquisto, della configurazione, e della manutenzione delle licenze o delle macchine necessarie per erogare i propri servizi. Sarà quindi compito del fornitore mantenere l'infrastruttura per gestire e distribuire i servizi, il quale verrà pagato in base al servizio offerto, solitamente attraverso un canone mensile in base al consumo.
Grazie a questa soluzione, le aziende godono di importanti vantaggi :

- **Riduzione degli investimenti iniziali**: pago solo quello che uso. All'aumentare del mio business, aumento gradualmente la richiesta del servizio, evitando grossi e rischiosi investimenti iniziali.
- **Maggiore libertà per lo staff IT**, il quale si affida al servizio senza doverlo gestire.
- **Riduzione del personale** addetto alla manutenzione e configurazione dei sistemi.
- **Affidabilità del servizio**, il fornitore ha come priorità la qualità del Cloud offerto, essendo esso stesso il suo business.

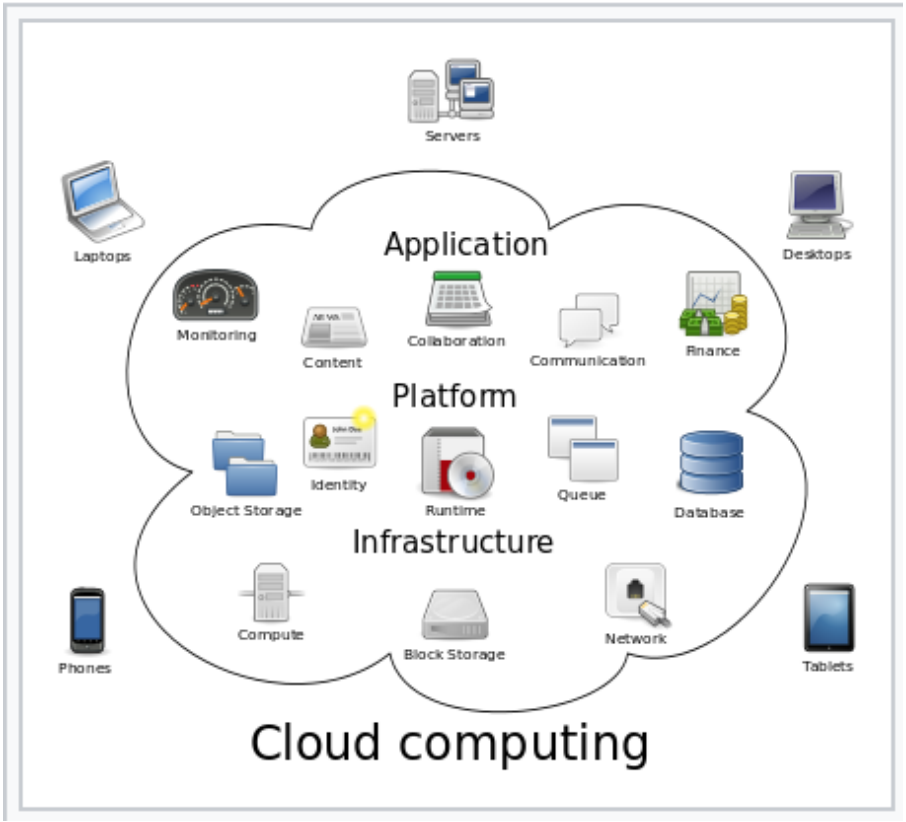

Figura 3.5: Schema logico dei servizi cloud. Fonte: wikipedia.com

3.3.2 Servizi Cloud

Il termine *Cloud Computing* è utilizzato in diversi contesti perché racchiude sotto di sé diverse tipologie di servizi :

1. **Saas (Software as a Service)**: con questo termine si intende la distribuzione di diverse tipologie di software (gestionali, posta elettronica, CRM) che risiedono su server remoto (dell'azienda che li distribuisce) attraverso la rete Internet. Gli utenti e le aziende che comprano il servizio possono accedere ad esso attraverso un'interfaccia web.

2. **PaaS (Platform as a Service)**: Questo servizio consente di fornire all'azienda che lo utilizza un intera piattaforma che supporta lo sviluppo di applicazioni in cloud computing. La piattaforma contiene tutto il necessario per sviluppare codice, come i linguaggi di programmazione, le librerie e i servizi che il provider vuole offrire.
   Una PaaS permette a chi la utilizza di non farsi carico dell'acquisto del software e delle architetture per poter sviluppare applicazioni, ma utilizzare un ambiente già configurato per lo sviluppo, il debug e le diverse analisi utili al programmatore. Il tutto tramite una interfaccia *web-based*, a cui si può fare accesso grazie a un semplice browser. Questo comporta una notevole semplificazione un grosso risparmio di tempo, evitando tutte le problematiche relative alla configurazione e aggiornamento degli ambienti di sviluppo. Lo sviluppato deve semplicemente accedere a internet e focalizzarsi sulla creazione del prodotto.

3. **DaaS (Data as a Service)**: vengono messi a disposizione del cliente dati, accedibili attraverso qualsiasi tipo di applicazione, come se fossero sul disco locale dell'utente. Molte volte le PaaS hanno integrati al loro interno sistemi di DaaS o DPaas (Data Platform as a Service) che permettono alle aziende di sviluppare software che utilizza infrastrutture dati gestite dallo stesso provider.

4. **IaaS (Infrastructure as a Service)**: Il termine indica l'outsourcing più esteso, che comprende tutte le risorse ICT. Il cliente che utilizza lo IaaS, affida al provider una serie di infrastrutture che costituiscono un intero data center virtuale. Si hanno a disposizione risorse hardware, reti e infrastrutture relative, sistemi di memoria per archivio e backup.

3.3.3 Principali piattaforme cloud

I principali fornitori di servizi Cloud sono, come prevedibile, le grandi aziende informatiche della Silicon Valley. Fra i servizi principali troviamo infatti *Amazon Web Service* (AWS), *Google Cloud Platform* (GCP) e *Microsoft Azure*. Queste piattaforme permettono di acquistare diversi servizi, e scalare le infrastrutture in funzione del traffico.

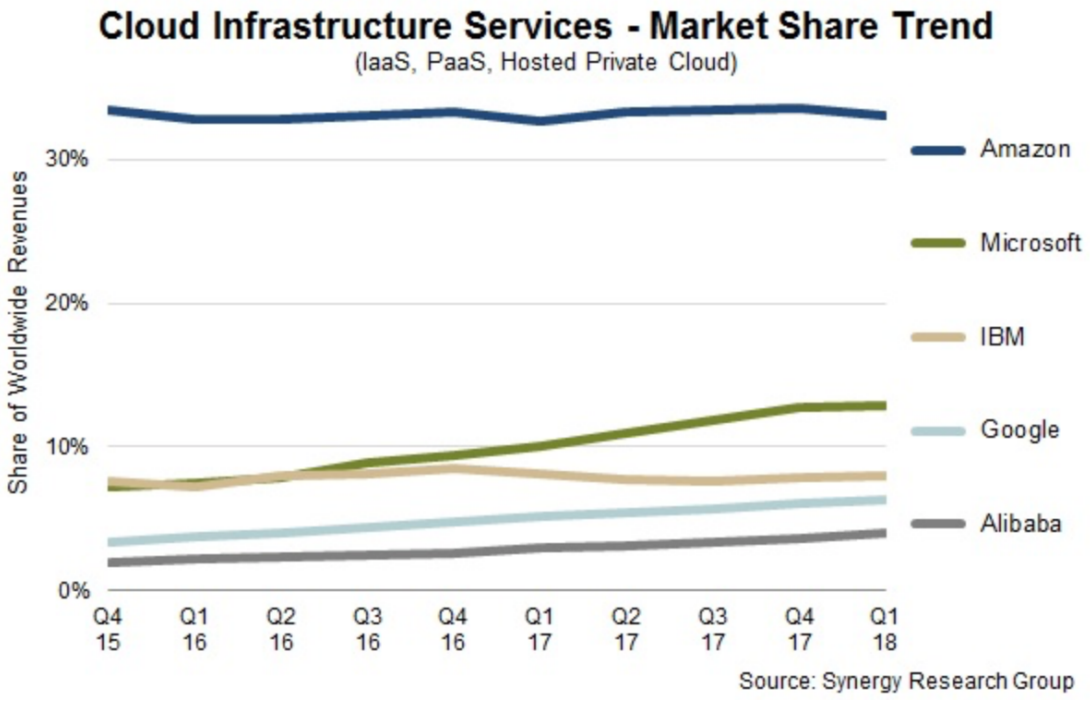

Figura 3.6: Divisione del mercato fra principali piattaforme cloud.

Come si può vedere da grafico in figura 3.6, Amazon primeggia in questo settore, occupando da solo più del 30% del mercato. Google è arrivata in ritardo, ma con i nuovi servizi come *Firebase* e *App Engine* e la possibilità di integrare tool di sviluppo e infrastrutture hardware, si sta ritagliando uno spazio importante.

Capitolo 4

Il framework Instant Developer Cloud

Il framework che ho utilizzato per sviluppare l'applicazione è *Instant Developer Cloud* (IDC), di proprietà di *Pro Gamma S.p.a*, una azienda informatica italiana.
IDC è una piattaforma di sviluppo per la creazione di soluzioni web e applicazioni *mobile* multipiattaforma, basata su standard web, permette di lavorare in Javascript, HTML5 e CSS. Consente di realizzare *front-end*, *back-end*, strutturare l'architettura dei dati e gestire la pubblicazione nei diversi app store con semplicità.
Il prodotto, rilasciato nel 2015, si sta diffondendo e ad oggi ha già permesso a diverse compagnie informatiche, agenzie web, startup e sviluppatori freelance di sfruttare la semplicità e la velocità che la piattaforma offre per creare prodotti multipiattaforma nel Cloud. Nel 2017 è stato selezionato da Google Cloud come *Technology Partner* della piattaforma GCP (Google Cloud Platform).
In questo capitolo parlerò di IDC analizzandone le funzionalità e l'architettura, illustrando i framework integrati al suo interno.

4.0.1 Architettura del framework

Facendo fede alle definizioni illustrate nella sezione 3.3.2, possiamo collocare IDC nella categoria dei PaaS, in quanto all'interno dell'ambiente messo a disposizione, è possibile programmare codice per le applicazioni, creare una interfaccia grafica, utilizzare librerie javascript, effettuare il

debug del codice, gestire i dati, analizzare le performance e pubblicare le applicazioni sugli store. Il framework infatti è nato per offrire un ambiente di sviluppo completo "in Cloud".
L'ambiente è già configurato e basta accedere alla *web-app* per poter programmare attraverso l'IDE messo a disposizione.
Volendo entrare nel dettaglio, IDC sarebbe definibile come una aPaaS (application PaaS), in quanto è una applicazione basata su *Google Cloud Platform*.
IDC utilizza ed estende le funzionalità di GCP permettendo di utilizzarlo in modo più semplice e prevedibile. Non possiede quindi una propria infrastruttura, ma si appoggia a quella di Google personalizzandone la presentazione.
La logica che permette alle app di essere eseguite sulle diverse piattaforme è basata su *Apache Cordova*, che è utilizzato in automatico dal framework quando deve distribuire e testare l'applicazione per Android o iOS. Il programmatore quindi non deve preoccuparsi di installare l'ambiente Cordova, perché i plug-in utilizzabili sono già messi a disposizione in maniera intuitiva dall'IDE.

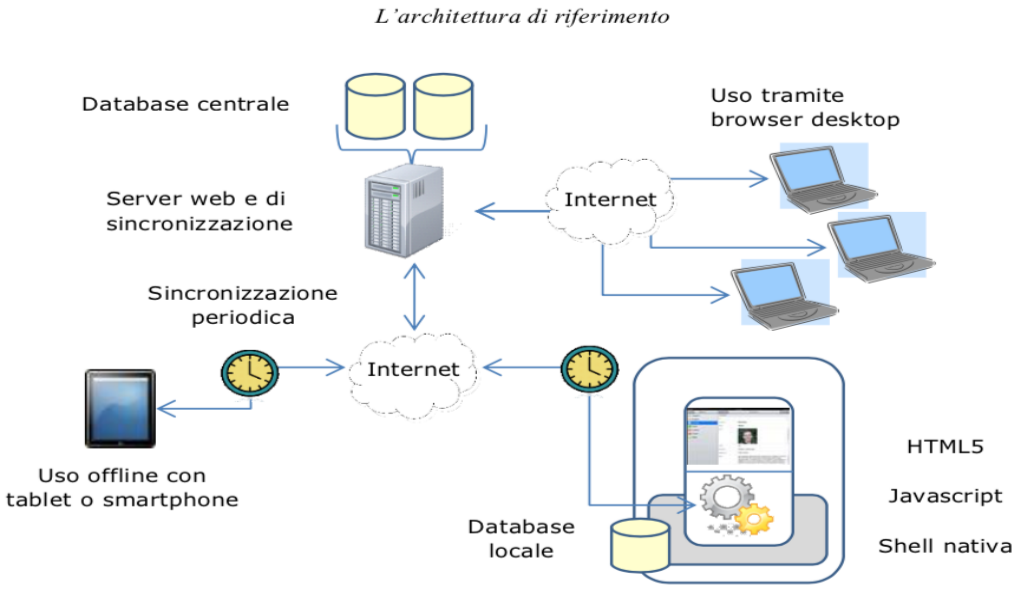

Figura 4.1: Schema logico funzionamento Instant Developer Cloud Fonte: InstantDeveloperCloud

Tecnicamente l'architettura di una applicazione mobile in IDC è composta da un server centrale e dalle applicazioni distribuite.
Il server centrale fornisce l'accesso ai dati, come per una normale applicazione internet, e permette la sincronizzazione dei database locali dei dispositivi, che servono per il funizonamento offline.
L'applicazione invece è costituita da un contenitore nativo, gestito da *Cordova*, che mette a disposizione i servizi del device grazie ai suoi plug-in, e dall'applicazione sviluppata, scritta a in HTML5 e Javascript, che implementa le funzionalità. La parte nativa non cambia mai, mentre l'applicazione web al suo interno viene aggiornata ad ogni modifica.
Una caratteristica molto utile del Cloud è quella della condivisione del codice all'interno del proprio team di sviluppo. Un sistema di controllo delle dipendenze permette il salvataggio automatico ad ogni modifica, la possibilità di effettuare *commit* del codice, creare *branch* ed effettuare i *merge* (seguendo la logica di Git).
Il codice è quindi mantenuto sul Cloud, e non bisogna preoccuparmi di gestire i salvataggi in locale. Il Cloud permette comunque di effettuare backup e scaricare il codice sorgente, che è sempre a completa disposizione.

4.1 IDE e struttura dell'app

4.1.1 IDE

L'IDE (Integrated Development Enviroment) di IDC è molto semplice e intuitivo da usare. Tutto l'ambiente è stato pensato per favorire lo sviluppo ed aumentare l'efficienza, in termini di tempo, di applicazioni multipiattaforma.
L'ambiente di sviluppo permette di accedere alla sezione personale attraverso le proprie credenziali, in cui sono presenti i progetti personali in fase di sviluppo, con una serie di informazioni tecniche proprie di ogni progetto. I progetti sono eseguiti su server che si possono gestire all'interno del Cloud, i quali variano la loro disponibilità in base alla licenza che si decide di acquistare, tipico dei servizi PaaS.
Aprendo l'editor di un progetto, viene presentato l'ambiente di sviluppo, il quale è diviso in cinque sezioni principali :

- **Dashboard**: questa finestra offre una panoramica del progetto, e permette la creazione di più istanze di applicazioni, per differenziarle in base al dispositivo principale su cui verranno eseguite e personalizzare la loro *user interface*, facendo riferimento sempre allo stesso *back-end*. In questa sezione è anche possibile gestire la condivisione del codice con la propria "organizzazione" di lavoro, composta da altri sviluppatori o da clienti finali, ed eseguire i vari commit, branch e merge del codice in fase di sviluppo.

- **Applicazione**: questa sezione è quella dedicata al front-end. Qua si possono gestire e creare le varie videate che compongono l'app. Si possono disegnare le interfacce selezionando gli oggetti direttamente nei menù di costruzione, o editare il codice HTML e CSS. Per ogni oggetto utilizzato, sia esso un semplice button o una videata, sono disponibili una lista di proprietà, la gestione dello stile, la lista degli eventi, l'insieme di elementi inseribili, le animazioni e la documentazione, il tutto senza dover accedere al codice.

- **Datamodel**: permette la gestione dei dati e la creazione dei database. L'interfaccia è semplice e intuitiva e consente di gestire molti aspetti senza scrivere codice. Uno strumento utile è *SQL view*, un editor studiato apposta per la scrittura di query SQL pre-impostate. I record risultanti da queste interrogazioni verrano mostrati in tabelle, direttamente modificabili.

- **Librerie**: questa sezione contiene il back-end. Qua sono presenti le librerie che vengono pre caricate automaticamente quando si crea un nuovo progetto. Librerie contiene anche le classi del framework DO, del quale parlerò nella sezione 4.3.2, che corrispondo ai documenti che si sceglie di creare.

- **Pacchetti**: sezione dedicata all'importazione di pacchetti esterni o esportazione di pacchetti personali interni. Infatti IDC permette di utilizzare pacchetti al di fuori di quelli pre caricati nel progetto, come quelli per la gestione dell' *user interface*.

4.1.2 Videate

Analizziamo adesso la struttura principale di una applicazione e quali sono le proprietà fondamentali che consentono l'accesso alle funzionalità.
Ogni applicazione è composta da videate, dette *View* che non sono altro che classi che vengono create nella sezione di sviluppo *front-end*. Come classi, esse hanno metodi e proprietà, ma anche una serie di elementi visuali che ne compongono l'interfaccia e in grado di notificare eventi.
Gli eventi sono inseriti all'interno dell'oggetto a cui si vogliono associare. L'evento più utilizzato è sicuramente l' *onLoad* sulle *View*, responsabile del caricamento della videata all'attivazione di essa.
Ogni View, può essere utilizzata come oggetto grafico, e quindi può essere aggiunta nella visualizzazione di altre interfacce, se viene attivato il flag *design time* su di essa.
Per passare da una videata all'altra occorre chiamare il metodo *show* sull'oggetto App, il quale rappresenta l'intera applicazione, oppure utilizzare librerie esterne che gestiscono il passaggio da una *View* all'altra secondo le modalità native del dispositivo su cui sono in esecuzione, come la classe *Pages* di Ionic.

4.1.3 Librerie Classi e proprietà

Quando si crea un progetto in IDC vengono caricate automaticamente due librerie che sono fondamentali per l'esecuzione dell'app multipiattaforma, e permettono di gestire le varie funzionalità del device in uso.

- **ApplicationLibrary**. Questa libreria consente di accedere alle funzionalità del dispositivo su cui verrà eseguita l'applicazione, mediante il framework Cordova.
  Essa contiene alcune classi fra cui *App* e *Device*, le più utilizzate. La Classe *App* rappresenta l'intera applicazione e, attraverso le sue proprietà e i suoi metodi, permette di gestire le funzionalità dell'app che non dipendono strettamente dal dispositivo. Alcune proprietà sono *views* per le videate, *sessionName* per le informazioni di sessione, *sync* per la sincronizzazione con il Cloud e *theme*. I metodi gestiscono le proprietà e la creazione di popup, nelle diverse forme (alert, confirm ecc. . .).
  L'oggetto *app* (con la a minuscola) rappresenta invece la sessione di lavoro dell'utente ed è l'istanza della classe *App*.

Si utilizza quindi *app* per riferirsi alla sessione, mentre *App* quando si vuole accedere ad oggetti che vivono in un contesto statico, come l'istanza di default delle *View* .

```
1     app.alert("messaggio");
2     App.Videata1.show();
```

Listato 4.1: Esempio di utilizzo classe App e oggetto app

Un'altra classe importante di *ApplicationLibrary* è *Device*, le cui proprietà sono valorizzate in base al device in cui l'applicazione viene eseguita. Possiamo ottenere da queste proprietà informazioni sul tipo di device e il suo codice univoco, il sistema operativo, le dimensioni dello schermo, il livello della batteria.
I componenti fondamentali per l'interfaccia dell'app con il sistema nativo, si trovano nella cartella *Plugins* di questa libreria. Al suo interno troviamo infatti diverse classi che rappresentano i plugins, forniti principalmente da Cordova e PhoneGap, che si occupano di gestire i problemi di interfaccia con i sistemi operativi, e permettono al programmatore di utilizzare dei semplici oggetti, con i loro metodi.
Si trovano quindi tutte le funzionalità principiali che un'applicazione potrebbe utilizzare : *barcodeScanner, vibration, camera, accelerometer, geolocalization, notification, nfc, contacts, sms, touchID* e altri, ma anche plugin di terzi, come l'accesso con le credenziali Facebook o Linkedin.

- **DBibrary**,la libreria incaricata della gestione dei database. Essa contiene le classi per la gestione delle varie tipologie di database supportate, quali *SQLite, Postgres, Oracle, SQLServer e MySQL.*

A queste due librerie caricate di default dal framework, si possono aggiungere tutte le librerie che servono per gestire l'applicazione, e popolarle di classi con metodi e proprietà nella sintassi del linguaggio Javascript.

4.1.4 Chiamate asincrone

Il linguaggio Javascript, tra le sue carattistiche, permette di utilizzare la programmazione asincrona, che è una forma di programmazione parallela, che consente ad un'unità di lavoro di funzionare separatamente dal *thread* principale notificando la fine del lavoro.
Essa torna utile quando si sviluppano applicazioni che hanno una interfaccia con l'utente, e le azioni da eseguire seguono gli eventi che l'utente genera. L'idea è quella di non aspettare che un evento esegua le sue funzioni, ma di eseguirle in background, con un nuovo thread a cui si passano delle *callback*, e restituire così immediatamente il controllo dell'interfaccia all'utente.
Questa forma di programmazione complica il codice e IDC permette di utilizzare un meccanismo automatico di sequenzializzazione delle operazioni. Sono disponibili due tipologie di comportamenti da seguire in seguito alla chiamata di un metodo asincrono :

1. Utilizzo della keyword *yield* davanti al metodo asincrono. L'editor di IDC inserirà in automatico la keyword ogni volta che viene chiamato un metodo asincrono. *Yield* permette di sincronizzarsi con il metodo e aspettare che esso ritorni al flusso principale.
   Risulta utile ad esempio usare *yield* per mostrare i popup che richiedono una conferma da parte dell'utente.

2. In alcuni casi è utile far proseguire il codice di un metodo senza aspettare il completamento delle operazioni asincrone che esso esegue. E' possibile rimuovere quindi la keyword *yield* e l'editor aggiungerà automaticamente alla chiamata il parametro per inserire la *callback*, ovvero la funzione che viene eseguita quando il metodo asincrono ritorna.

4.2 Front-end e User Experience

4.2.1 Gestione del front-end

Un aspetto molto curato da IDC è la creazione del *front-end*. Il framework propone una soluzione molto rapida e intuitiva, che permette di disegnare le videate direttamente nell'IDE . Essendo il sistema *web-based*, i linguaggi utilizzati per creare elementi grafici sono HTML e CSS. I componenti delle videate si possono quindi creare utilizzando questi linguaggi, ma l'IDE mette a disposizione una serie di template e componenti già pronti. I componenti si possono aggiungere e andare a personalizzare a piacere, essendo sempre presente il codice nativo che li genera.
Un aspetto in evoluzione delle attuali tecnologie che offrono applicazioni multipiattaforma è quello di portare la *user experience* al livello delle applicazioni native, in termini di componenti visuali e performance. L'obiettivo è quello di non fare percepire all'utente finale quando si sta utilizzando una applicazione nativa o una *cross-platform*.
Per riuscire a risolvere questo problema, si sono sviluppati diversi framework che contengono numerosi elementi necessari per comporre le videate secondo i canoni estetici richiesti dalle app odierne.
Per fare qualche esempio si possono citare *Ionic*, che ho utilizzato all'interno del mio progetto, *Material Design* di Google, *Onsen* e molti altri.
Questi framework permetto di velocizzare la creazione delle view, ovvero le schermate che compongono le applicazioni, mettendo a disposizione componenti, gestures, animazioni, icone, colori e temi per creare una *user interface* di livello. Inoltre gestiscono autonomamente l'interfaccia con il dispositivo su cui vengono eseguiti a livello di sistema operativo e device. Infatti questi framework sono in grado di mostrare i componenti nativi, come pop-up, liste, buttons, in funzione del sistema operativo su cui vengono eseguiti e adattare le dimensioni se il dispositivo è uno smartphone o un tablet, orizzontale o verticale.
Per fare questo, al programmatore della UI, sarà richiesto di programmare una sola interfaccia, che si adatterà automaticamente. IDC permette di importare queste librerie, ed utilizzare i loro componenti in base alle specifiche del progetto, oppure utilizzare una serie di elementi già configurati, presenti all'interno del framework.

4.2.2 Framework IONIC

Ionic Framework è uno dei più diffusi strumenti per lo sviluppo di app multipiattaforma, il quale, programmando una sola volta, utilizzando javascript, CSS3 e Angular, permette di sviluppare la stessa app per Android, iOS e Windows.
Il framework è *open-source* ed è *front-end oriented* in quanto è focalizzato sul design della *user interface* dell'app.
Ionic utilizza il pattern architetturale MVC (*Model View Controller*), molto utilizzato nella programmazione a oggetti delle applicazioni web, che separa la logica di presentazione dalla logica di controllo.
Ionic è scritto in Javascript, ma sfrutta la potenza e le prestazioni di Angular e Node.js. Lo strato di basso livello si interfaccia con il sistema operativo utilizzando Cordova, grazie al quale ottiene l'accesso alle diverse componenti del dispositivo, come fotocamera, GPS, accelerometro ecc..
Gli sviluppiatori *front-end* e web designer, non sempre sono in grado di creare applicazioni utilizzando i linguaggi nativi (Java per Android, Swift per iOS). Ionic permette invece di creare applicazioni per ogni piattaforma anche a queste figure, che molto probabilmente conoscono Javascript e CSS.
L'aspetto su cui fa leva oggi Ionic è la potenza dei dispositivi in commercio, soprattutto quelli di ultima generazione: nonostante il caricamento di un'app multipiattaforma può essere lento rispetto ad una nativa, i componenti hardware sempre più performanti permettono di eliminare la percezione della differenza di prestazioni nel rendering grafico, permettendo ai framework come Ionic di ritagliarsi uno spazio importante nel mercato.

4.3 Back-end

4.3.1 ORM

Per il *back-end* di IDC, è previsto un framework ORM che permette di evitare di comporre molte query, scrivendo al loro posto codice javascript che lavora secondo le specifiche ORM, velocizzando il processo di creazione del back-end.
In Informatica, l'*Object-Relational Mapping* è una tecnica di programmazione, che favorisce l'interazione di sistemi software aderenti al paradigma della programmazione orientata agli oggetti, con sistemi RDBMS [15].

ORM si riferisce quindi alla tecnica di mappatura dei dati tra modelli e linguaggi object oriented e modelli di dati relazionali.
IDC fornisce un suo sistema ORM per poter gestire i record dal database in modo strutturato e notevolmente più veloce. il framework che usa IDC è chiamato DO, che significa *Document Orientation*.
Il framework DO permette le operazioni sui dati del database senza la scrittura di query SQL creando un livello di astrazione dovuto dalla mappatura delle tabelle di un database con classi Javascript. La scrittura di query SQL, che a volte richiede una significante parte del tempo utilizzato per sviluppare un applicazione, viene così interamente gestita dal framework DO, il quale, a seconda dell'utilizzo e delle impostazioni che verranno assegnate al singolo documento, si occuperà di effettuare le query necessarie, in maniera indipendente.
E' comunque sempre a disposizione la possibilità di utilizzare le query in maniera "classica", procedura che ritorna utile quando le relazioni del database hanno una struttura particolarmente complicata.

4.3.2 I Documenti

Le classi in cui vengono mappate le tabelle dei database, sono chiamate Documenti. Il Documento, nel framework DO, è un particolare tipo di classe il cui compito è quello di gestire i record all'interno del database. Una istanza delle classi Documenti è chiamata Documento. Il documento è la rappresentazione nel linguaggio *object oriented* di una singola tupla di una relazione del database.
E' possibile effettuare sui documenti le operazioni fondamentali (CRUD) applicabili su una risorsa :

- **C = Create**: la creazione di un documento equivale alla istanza di un oggetto in Javascript. Utilizzando la keyword *new*, viene istanziato un oggetto della classe Documento interessata. Possono essere passati i campi eventualmente noti relativi al documento da istanziare come parametri. Se non viene indicato alcun valore per i campi "*not null*", il procedimento di salvataggio del documento fornirà risultato negativo. Importante, nella fase di Create, è la proprietà *inserted*, da impostare a true per notificare la nuova creazione al framework, il quale dovrà inserirlo nel database.

- **R = Read**: La lettura di una documento è certamente l'operazione più comune. Se si vuole ottenere un singolo documento (relativo ad una singola tupla di una relazione), si utilizza il metodo *loadByKey*, al quale si passa la chiave primaria del record richiesto. Se invece si intende leggere una collezione di dati, si utilizza il metodo *loadCollection*, che permette di specificare eventuali filtri di caricamento. Il metodo ritorna una lista di documenti che può essere salvata in un oggetto di tipo *Collection*. La classe *Collection* è pensata appunto per gestire liste di documenti come semplici vettori.
- **U = Update**: la modifica di un documento è estremamente semplice, in quanto una volta caricato in un oggetto, con i metodi discussi precedentemente, si può accedere alle proprietà e modificarne il valore a piacere.
- **D = Delete**: si utilizza la proprietà *deleted* impostandola a *true* sull'oggetto da eliminare.

Si noti l'efficienza delle procedure appena analizzate : è sufficiente chiamare un singolo metodo o impostare il valore di una proprietà, invece di comporre una query, per compiere azioni sui dati all'interno del database. I dati saranno comunque manipolati da query, che però saranno composte dal framework DO in background, inserendo e parametri necessari.
Esistono alcuni metodi fondamentali che servono per indicare al framework di allineare i documenti manipolati con il database di riferimento su cui verranno effettuate le query.
Uno fra tutti è *save*. Questo metodo, chiamato su un documento, lo allinea al database e restituisce *true* o *false* a seconda che sia andato a buon fine o meno.
Un altro metodo molto utile è "restoreOriginal", il quale viene chiamato su un documento a cui vogliamo annullare le modifiche. Questo metodo riporta il documento allo stato originale, cioè a quando è stato caricato dal database. Se il documento è stato *inserted*, lo rende anche *deleted*.

I Documenti sono salvati una cartella personale del *back-end*, creata sotto la sezione "Librerie" e ad ognuno di essi saranno associate diverse sottocartelle per contenere le relative risorse, le quali possono essere :

- **Properties**: sono le proprietà del documento in questione. che derivano dai campi della tabella del database a cui fanno riferimento.

L'accesso ad ad esse è del tutto identico a quello di normali proprietà di un oggetto (oggetto.proprietà).
Alle Properties "originali", quelle derivate direttamente dalla tabella, possiamo aggiungere properties "derivate", proprie di tabelle in relazione con la principale, collegate da chiavi esterne del modello relazionale. Infine è possibile aggiungere ai documenti proprietà non legate ad alcun campo del database. Queste proprietà vengono chiamate *Unbound*, che contengono informazioni utili per la gestione del documento (come somme totali o flag di controllo).

- **Collection**: le collection sono particolari tipi di proprietà che contengono documenti "figli" appartenenti ad una tabella che è in relazione con la tabella del documento "padre". Quindi data un'istanza del documento padre, la sua collection conterrà solo le istanze del documento figlio a lui collegate. Questa funzione è molto utile per evitare di scrivere query che richiedono di mettere in join più tabelle.
  La logica può essere eseguita anche sui documenti delle tabelle "figlie" creando così una catena di dipendenze. I metodi per la gestione dei documenti, in particolare quelli di caricamento, si adattano a questa caratteristica accettando dei parametri relativi al *childLevel* e caricando i documenti figli scendendo alla profondità indicata. Il metodo *save*, salva automaticamente tutte le modifiche che sono effettuate su ogni livello di parentela del documento sul quale viene chiamato.

- **Metodi**: E' possibile aggiungere metodi e funzioni ad ogni documento. Essi costituiscono il cuore del *back-end* della nostra applicazione. Grazie ai metodi andrò a gestire la logica dell'applicazione, scrivendo le istruzioni da eseguire sui dati e sulle azioni compiute su di essi. I metodi hanno 3 modificatori di accesso (public, protected, private) che consentono di gestire la visibilità nel caso le librerie di *back-end* vengano esportate ed usati in altri progetti.

- **Eventi**: Per ogni documento posso andare ad intercettare numerosi eventi per lavorare sui dati a diversi livelli. Gli eventi sono innestati dai metodi chiamati sui documenti e dalle modifiche delle proprietà. Ad esempio, chiamando il metodo *save*, viene lanciato l'evento *onSave*, all'interno del quale posso fare un controllo di validità dei dati.
  Un altro esempio è l'evento *onDeleting*, notificato quando cambia la

proprietà *deleted*. Altri eventi importanti sono *after/beforeLoad* piuttosto che *onUpdating*, utili quando bisogna aggiornare dati (come le proprietà unbound) prima o dopo il caricamento o la modifica.

- **Query**: Come accennato in precedenza, è sempre possibile caricare un documento di dati risultanti da una query scritta ad hoc per una particolare esigenza. Per fare questo occorre aggiungere al documento un oggetto "query", il quale predispone un'interfaccia per scrivere query strutturate. I campi risultanti dell'interrogazione (quelli indicati nella clausola select) verranno trattati come properties del documento.

4.3.3 DataMap per collegare front-end con back-end

Instant Developer Cloud propone una soluzione molto intuitiva e semplice da utilizzare per collegare *front-end* e *back-end*.
Il framework si serve di alcuni standard per la comunicazione in rete, come *WebSocket* e *WebRTC*, e li utilizza in maniera automatica, permettendo allo sviluppatore di evitare di affrontarne la complessità. Il perno sui cui fa leva IDC, è il fatto che lo sviluppo di applicazioni, che siano esse mobile o servizi web generici, comporta una serie procedure di sviluppo da seguire che nella maggior parte dei casi, si ripete, e vede il programmatore risolvere indicativamente sempre lo stesso tipo di problema.
Ad esempio, in una applicazione *data-driven*, si andranno principalmente a fare le operazioni base sui dati (CRUD) e poi si chiederà di salvare le modifiche in un database. Per velocizzare questo processo, ed evitare di creare numerosi WebService, che richiederebbero tempo, IDC propone la sua soluzione : le DataMap.
Le DataMap sono i componenti creati apposta per collegare il *front-end* al *back-end* in maniera rapida e intuitiva. Attraverso le DataMap è possibile collegare direttamente gli elementi visuali delle videate (come campi di input, lables, selezioni multiple ecc..) alle proprietà dei documenti.
In seguito sono illustrati i principali passaggi che son necessari per effettuare il collegamento indicato :

1. Aggiungere un elemento DataMap alla *View* interessata. L'oggetto sarà disponibile solo all'interno della *View* indicata e potrà essere caricato per fornire dati relativi ad essa.

2. Dopo aver aggiunto la DataMap si indica quali documenti dovrà gestire. Per fare questo basta indicare il documento scelto all'interno della proprietà "Tipo contenuto".
   Automaticamente alla DataMap vengono aggiunte tutte le proprietà del documento, alle quali, da ora in poi, potrò fare accesso utilizzando l'oggetto DataMap ($datamap.properties).

3. In seguito è necessario indicare dove verranno visualizzati i dati all'interno della videata. La DataMap possiede una proprietà *template*, molto utile quando bisogna associare la DM a una lista di elementi. Il *template* è l'elemento visuale che verrà usato come modello per ogni documento da mostrare nella lista, il quale a sua volta può contenere diversi elementi grafici. (Ad esempio il template è l'item di una listItems, e l'item contiene a sua volta diverse labels, ognuna associata ad una proprietà del documento).

4. E' necessario decidere come utilizzare le proprietà di ogni documento, assegnandole ai componenti visuali opportuni. Ogni elemento grafico possiede una proprietà "Sorgente dati", che andrà collegata alla proprietà della DataMap a cui è associata, e una "Proprietà collegata", che indica quale attributo dell'elemento deve essere associato alla proprietà : ad esempio *value* per un inputText, oppure *innerText* per una label.

5. Infine è necessario indicare quando caricare i dati. Solitamente questo viene fatto all'apertura di una nuova videata, quindi sull'evento *onLoad* della *View* viene chiamato il metodo *load* sulla DataMap.

6. E' importante citare per ultimo l'evento *onRowComposition*, che torna utile quando è necessario adattare l'aspetto grafico dell'elemento da visualizzare in funzione dei dati dello stesso.
   Ad esempio nella composizione di una lista, se ogni riga deve avere un aspetto personalizzato in funzione dei suoi valori, possiamo utilizzare questo metodo per gestire le modifiche.

E' notevole quindi il vantaggio portato dalle DataMap. Senza di esse, sarebbe più laborioso andare a gestire le modifiche e la visualizzazione dei dati nelle diverse pagine. Ogni properties di ogni documento dovrebbe caricata, modificata e salvata a se, all'occorrere di uno specifico evento,

come il click su un bottone. Attraverso le DataMap invece abbiamo un collegamento bidirezionale fra la properties del documento e l'elemento visuale.
Per fare un esempio, possiamo pensare di collegare un campo di input ad una proprietà del documento attraverso una DataMap, con i metodi visti sopra. Quando l'utente modifica i valori del campo, anche la proprietà collegata assumerà un nuovo valore.
In seguito sono analizzati alcuni strumenti utili applicabili alle DataMap

Filtri
In fase di caricamento, prima di utilizzare il metodo *load*, si possono aggiungere alle DataMaps una serie di filtri, che vanno ad agire sui dati dei documenti caricati, escludendo dal caricamento le istanze dei documenti che non rispettano i parametri richiesti. I filtri possono essere concatenati a piacere e permettono di agire su diversi attributi delle proprietà, come uguaglianze, intervalli, confronti tra date e espressioni di altro genere.

Paginazione
Le DataMap hanno la possibilità di utilizzare la tecnica della paginazione. Quando i dati che devono essere caricati all'interno di una DataMap sono numerosi, se caricati e mostrati tutti in una sola volta, posso rallentare le prestazioni dell'applicazione. La paginazione permette appunto di risolvere questo problema. Inizialmente verrà mostrato solo un sottoinsieme delle righe, e man mano che si scorrerà la lista, verranno caricati gli elementi successivi (o precedenti).
Per attivare la paginazione è sufficiente impostare la proprietà *pageSize* della DataMap che indica la capienza di elementi da caricare all'interno di ogni pagina. Attraverso l'evento *onNextPage*, notificato quando la DataMap sta per aggiungere una nuova pagina di dati, è possibile modificare il numero delle nuove righe da aggiungere alla pagina, gestendo la proprietà *event.increment*.

Query
Come per il framework DO per la gestione dei documenti, anche per le DataMap è consentito l'utilizzo di query personalizzate per il caricamento dei dati all'interno della struttura. Per fare questo occorre aggiungere alla DataMap un oggetto query, che verrà utilizzata come sorgente dati.

Innestate
Ci sono casi piuttosto comuni in cui è necessario mostrare i dati di contesto relativi a una riga di una lista, come ad esempio può avvenire per un ordine e il suo dettaglio. Per fare questo si utilizzano le DataMap innestate, che non sono altro che DataMap contenute in altre DataMap.
Esse funzionano come DataMap normali, sono collegate a un certo documento, si possono applicare dei filtri e sono collegate a uno specifico template. E' necessario però che il template della DataMap innestata sia contenuto nel template della DataMap che la contiene. Come si può intuire, il caricamento di una DataMap innestata dovrà avvenire alla creazione di ogni riga di quella esterna, quindi si utilizzerà il metodo *load* all'interno dell'evento *onRowComposition* della DataMap di testata.

Campi di Input
Le DataMap sono utilizzate spesso quando si utilizzano campi di input di tipo *select*, ovvero quando l'utente ha la possibilità di scegliere tra una serie di valori disponibili, come le categorie di un certo prodotto, o i giorni della settimana.
Le opzioni che un campo di questo tipo deve mostrare solitamente sono lette da un database, come il resto dei dati. Quindi possiamo sfruttare la funzionalità delle DataMap per popolare queste select. Per fare questo è necessario aggiungere una DataMap direttamente all'interno dell'elemento visuale, e impostare le proprietà della datamap come spiegato in precedenza.

4.3.4 Collegamenti esterni

Questo paragrafo ha analizzato i principali aspetti che IDC mette a disposizione per gestire il *back-end* di una applicazione. Come si può intuire, se lo sviluppo avviene interamente all'interno del framework in maniera full-stack, si beneficia di tutti gli strumenti che esso mette a disposizione, dai documenti alle DataMaps.
Non sempre però i clienti hanno bisogno di sviluppare una applicazione *full-stack*, ma intendono collegarsi a un back-end già scritto e in esecuzione su altri server. IDC permette di interfacciarsi con servizi esterni, utilizzando le *WebAPI*.

4.4 Dati

4.4.1 Architettura del database

Instant Developer Cloud permette di definire l'organizzazione dei dati direttamente all'interno del framework, costruendo i database che serviranno all'applicazione per funzionare. Il Cloud permette di utilizzare diverse classi di database relazionali lato server, come *Postgres* (predefinito), *Oracle, SQLServer, MySQL*. Sui dispositivi invece viene utilizzato *SQLite*, grazie alle sue dimensioni ridotte, e utilizzato in molti casi anche da iOS e Android per le funzioni mobile.
Il Cloud di IDC dà la possibilità di creare una struttura dati partendo da zero. Si posso definire le varie tabelle, e gestire i collegamenti fra di loro utilizzando la logica delle relazioni dei modelli relazionali. A ogni tabella si possono aggiungere campi, ognuno con il suo "Tipo di dato", scegliendo da una lista di tipi comuni.
Le chiavi primarie delle relazioni possono essere create facilmente attivando sull'attributo il flag *primary key*, ed è gestita in automatico assegnazione del valore univoco in maniera *auto increment*, se il tipo della chiave primaria è impostato su "UUID".
Anche la creazione delle chiavi esterne è piuttosto semplice, in quanto basta aggiungere alla tabella un elemento *foreign key*, e associarlo all'attributo della tabella che vogliamo collegare. Per le chiavi esterne si possono poi selezionare i metodi di cancellazione o aggiornamento, propri dei database relazionali, come *Cascade* o *Restrict*.
Il framework IDC anche in questo caso quindi ci semplifica il lavoro, proponendo una sua interfaccia per la creazione di una struttura dati completa, cercando di abbassare la complessità e i tempi di sviluppo fornendo un utilizzo facilitato delle funzionalità più comuni per la creazione di un database.

4.4.2 Query

Vediamo ora quali sono gli strumenti che possiamo utilizzare per eseguire query all'interno delle nostre applicazioni. Come discusso nei paragrafi precedenti, IDC permette di evitare di scrivere molte query, che in molti casi possono essere autogenerate dal framework DO con l'utilizzo dei documenti, o dalle DataMap, le quali caricano i dati filtrati generando query

automatiche.
Non sempre però chi sviluppa l'applicazione riesce a sfruttare questi strumenti, trovandosi a gestire casi in cui i dati richiesti sono molto specifici e dipendono da molte tabelle, in relazione tra loro. Per questo è sempre possibile scrivere query per esteso, utilizzando l'editor messo a disposizione da IDC.
Una query infatti potrà essere scritta in qualsiasi funzione, sia nel *front-end* che nel *back-end*, inserendola in mezzo al codice javascript del flusso di esecuzione.
Per scrivere una query infatti è sufficiente chiamare il metodo "query" sul nostro database, ottenuto grazie alla classe *App*. Il metodo accetta come parametro una stringa, che equivale alla query da effettuare sulle tabelle. Quando si chiama questo metodo, l'IDE apre automaticamente un editor che permette di scrivere la query in maniera "strutturata", forzando la scrittura in maniera corretta. La query strutturata verrà tradotta automaticamente in base al tipo di database.
All'interno dell'editor inoltre è possibile utilizzare direttamente le variabili di contesto e interrogare il database in funzione dei dati applicativi.
Il metodo *query* ritorna il risultato in un oggetto che si intercetta in una variabile. Sarà sufficiente utilizzare il metodo *rows* su questa variabile per accedere ai record letti come se fossero in un comune vettore, applicando dei cicli per leggere tutte le tuple e indicizzando le righe con le parentesi quadre (risultato.rows[0]). Le query strutturate sono quindi indipendenti dal database su cui verranno eseguite, perché sono tradotte in automatico.
E' però prevista la possibilità di scrivere codice SQL personalizzato, in funzione della classe di database che utilizzo. Ad esempio se viene utilizzato Postgres, si possono sfruttare le sue funzioni di ricerca *full-text* per la ricerca del testo. In questo caso però, la query perde la sua proprietà "strutturata" e non è più garantita l'indipendenza dalla classe di database.

4.4.3 Transazioni

Il framework mette a disposizione la possibilità di eseguire in modo particolarmente efficiente e sicura gruppi di query attraverso l'uso delle transazioni.

Esse permettono infatti di gestire blocchi di operazioni SQL in modo atomico, garantendo che l'intero gruppo di istruzioni venga eseguito con successo. In caso contrario vengono annullate tutte le operazioni. Queste funzionalità sono molto utili quando le query da effettuare lavorano su dati importanti, come conti correnti o informazioni personali.
E' necessario avere la garanzia che un blocco di istruzioni sia eseguito correttamente, per evitare incosistenze nei dati che porterebbero problemi importanti, come la perdita di soldi su un conto.
Se non si specificano transazioni esplicite, esse sono eseguite automaticamente quando serve. In pratica le query vengono eseguite in modalità *auto-commit*, all'interno di una propria transazione, che viene aperta e chiusa autonomamente. Per gestire le transazioni in modo esplicito, occorre utilizzare tre metodi principali dell'oggetto database ottenuto da App:

1. **beginTransaction**: apre una transazione.

2. **execute**: esegue una query come il metodo *query* ma evitando l'*auto-commit*. Utilizzabile per tutte le istruzioni all'interno della transazione.

3. **commitTransaction**: chiude la transazione confermando le modifiche.

Per essere sicuri che la transazione verrà sempre chiusa, e quindi il flusso di esecuzione raggiungerà sempre il metodo "commitTransaction", è buona norma comprendere le istruzioni di execute all'interno di un costrutto *try-catch*, in mezzo ai metodi di apertura e chiusura, per evitare che l'errore su una execute non permetta di raggiungere il commit finale.
E' molto importante che le transazioni esplicite siano più brevi possibili in termini di tempo di esecuzione. Esse non devono dipendere quindi dal tempo di risposta dell'utente che potrebbe allungare il tempo della transazione in maniera rischiosa.
Infine, per annullare tutte le modifiche avvenute in una transazione è necessario utilizzare al posto del metodo *commitTransaction*, il metodo *rollbackTransaction*

4.4.4 Connector

Il Connector è lo strumento che offre IDC per connettere database esterni al Cloud. Non sempre chi sviluppa una applicazione intende creare un database nuovo, ma al contrario preferirebbe utilizzare i dati che possiede già all'interno dei database aziendali privati, e consentire ad essi l'accesso da un applicazione esterna.
Il "Cloud Connector" gestisce questo problema, nei casi in cui si voglia mantenere la logica del back-end sui server Cloud di IDC, mentre i dati su database privati.
Il Cloud Connector è tecnicamente un software, che si installa sulla macchina che ospita il database da pubblicare, e che consente accesso ai dati dal Cloud, senza dover aprire la rete all'esterno. Il software è basato su Node.js, ed è appunto un pacchetto di file javascript. Per utilizzarlo si deve istallare il framwork node.js sul server in cui è presente il database.
In seguito basterà inserire alcuni parametri nei file di configurazione, quali la lista dei server che il cloud deve contattare, su cui è eseguita l'applicazione di IDC, la lista degli utenti autorizzati e l'elenco dei datamodel da pubblicare attraverso il Cloud Connector. Il servizio supporta i database *Postgres*, *SQLServer* e *Oracle*.

4.5 Strumenti Utili e costi

IDC mette a disposizione altri strumenti utili per lo sviluppo e la pubblicazione di applicazioni, implementati all'interno dell'IDE di sviluppo, ma anche a livello di gestione e analisi dei progetti.

- **Pubblicazione**: IDC gestisce automaticamente la pubblicazione delle applicazioni su Apple Store e Google Play attraverso un Build Server appositamente creato. Inoltre è possibile effettuare dei *live update* che permettono di aggiornare le app senza ripubblicarle.
- **Test di carico**: permettono di effettuare dei test sulle applicazioni per avere riscontri sulla qualità del codice al crescere dei volumi di dati e di utenti.
- **Debug**: è possibile effettuare il debug in fase di sviluppo direttamente nel browser, il quale permette di visualizzare l'anteprima dell'app sulle varie tipologie di device virtuali: smartphone, tablet e desktop,

in modalità orizzontale o verticale.
In alternativa a questo metodo è possibile collegare un device fisico, sfruttando l'applicazione InstaLauncher di IDC su Android. Per i dispositivi iOS, visto che l'Apple Store non permette di caricare applicazioni per fini di test, è' disponibile l'applicazione *Test Flight*, che fornisce una piattaforma di debug, dove è possibile scaricare l'app *InstaLauncher*.

- **Offline**: è disponibile un framework per permettere il funzionamento delle applicazioni offline, ed avvicinarsi così ancora di più all'esperienza delle app native. Il framework lavora con i documenti, e non con i record del database. Attraverso la classe *Sync* permette di gestire lo scambio di dati fra l'applicazione sul dispositivo e il server sul Cloud, permettendo di utilizzare una sincronizzazione completa o differenziale.

- **Analitiche**: vengono registrate dal service le analitiche relative al comportamento dell'app in mano agli utenti, per fornire dati utili agli aggiornamenti possibili e le direzioni da prendere per migliorare l'esperienza e le prestazioni delle applicazioni.

- **Programmazione relazionale**: consiste in una nuova tecnica che si applica alla scrittura del codice, la quale memorizza le relazioni tra gli elementi del codice e le gestisce automaticamente nelle operazione di modifica. Questo sistema è molto utile perché permette di mantenere allineato il codice ed evitare errori molto frequenti dovuti a distrazioni o dimenticanze che fanno sprecare parecchio tempo a chi programma, nella fase di debug. Ad esempio, modificando il nome di un campo del database, la modifica verrà propagata al documento relativo, e a tutte le linee di codice che utilizzano quella proprietà.

Per utilizzare il framework Instant Developer Cloud è necessario comprare una licenza che varia a seconda del servizio richiesto sottoforma di canone mensile, come succede abitualmente per le piattaforme Cloud.
Gli abbonamenti acquistabili si differenziano in base al prodotto, sia esso un app per dipositivi mobili, o semplicemente un applicativo web accessibile da browser. Generalmente la licenza per utilizzare l'ambiente di sviluppo, per una sigola persona, è di 119€ al mese, a cui poi va aggiunta una somma di 29€ mensili per la pubblicazione sui vari *store*. Inoltre, qualora

si desideri utilizzare gli strumenti avanzati offerti dalla piattaforma, come analitiche, test di carico e servizi di backup, il canone aumenta di 69€ mensili, che permettono di sfruttare il sistema nella sua totalità.
Nonostante le cifre siano importanti, bisogna tenere in considerazione i costi di pubblicazione sui vari negozi virtuali, che sono compresi nei prezzi di IDC.
Una licenza molto utile per effettuare una prova del framework, è quella gratuita, che permette di avere piena accessibilità all'IDE, effettuare il debug delle app sui dispositivi fisici, e gestire i database. Essa limita però a 20 il numero di *View* aggiungibili al progetto, non permettendo la pubblicazione.

Capitolo 5

Sviluppo applicazione K-4Sales

5.1 Obiettivi

Questo capitolo ha lo scopo di analizzare le tecniche che ho utilizzato per sviluppare una applicazione full stack con il framework Instant Developer Cloud, che ho presentato in precedenza. Il lavoro di sviluppo dell'applicazione è stato svolto per conto dell'azienda *Kelyan S.p.a.*, nella sede di Savigliano, in provincia di Cuneo.
L'azienda fornisce un servizio Cloud per l'automazione di vendita attraverso una applicazione sviluppata per iOS in versione Tablet chiamata K-4Sales. Pertanto, l'applicazione è utilizzabile dai clienti solo su iPad.

5.1.1 Multipiattaforma e Multi-device

Il mio obiettivo principale è stato quello di adattare l'applicazione per portare il servizio su altre piattaforme e altri device, sviluppandone un prototipo.
La scelta è ricaduta fin da subito sull'utilizzo delle tecnologie cross-platform. K-4Sales è un servizio basato principalmente sulla gestione dei dati; gli agenti delle aziende che utilizzano il servizio, posso gestire i clienti e i loro ordini, automatizzando così il processo di vendita. L'applicazione infatti non presenta particolari requisiti da un punto di vista hardware e non utilizza componenti strettamente legati all'ambiente in cui viene eseguita.

Per questo, l'approccio multipiattaforma si sposa perfettamente con questo tipo di applicazione.
L'altro obiettivo di estensione è stato quello di sviluppare codice in modo che fosse disponbile per il funzionamento *multi-device*. Ecco che IDC risolve automaticamente questo requisito e con un solo *back-end* e qualche piccolo adattamento al front-end in funzione del dispositivo, ho potuto sviluppare un solo progetto estendibile allo sviluppo su smartphone, tablet e desktop.

5.1.2 Tempo di sviluppo

Secondo obiettivo è stato quello di prestare attenzione al tempo di sviluppo dell'applicazione, cercando di verificare se le tecnologie utilizzabili in IDC permettono davvero di creare un applicazione multipiattaforma abbattendo i tempi di sviluppo.

5.1.3 Sviluppo full-stack

Altro obiettivo del lavoro svolto è stato quello di utilizzare il framework nella sue piene potenzialità. L'intero sviluppo è stato effettuate all'interno del framework IDC, cercano di utilizzare tutti i principali strumenti, dalla gestione del database, al framework DO, fino alla creazione dell'interfaccia grafica.
Ho infatti scelto di sviluppare l'applicazione partendo da un nuovo progetto, senza riutilizzare il materiale che è già presente per la gestione dell'applicazione attualmente attiva, come il database o il *back-end*.

5.1.4 Flessibilità

L'applicazione K-4Sales permette di gestire una vasta gamma di clienti, ognuno con requisiti differenti. Possiamo quindi trovare un cliente che si accontenta delle funzionalità base dell'app, mentre altri che richiedono una personalizzazione maggiore del prodotto, in funzione dell'attività che devono svolgere e dei prodotti che vengono venduti. La sfida è quindi stata quella di creare una applicazione che potesse accettare queste customizzazioni in base alla tipologia di cliente.

5.2 Logica funzionale

In questo capitolo andrò ad analizzare il funzionamento del prototipo dell'applicazione K-4Sales che ho sviluppato.
L'applicazione orginale è stata sviluppata in parecchi anni e presenta al giorno d'oggi una vasta gamma di servizi, che i clienti acquistano per avere ulteriori informazioni sullo stato dei loro clienti e relativi ordini.
Nel prototipo sviluppato per conto di questa tesi, mi sono concentrato sulle funzionalità principali andando a ricreare logica centrale dell'app, senza soffermarmi sull'aggiunta dei servizi secondari.
L'*user experience* dell'applicazione segue quella dell'app originale ed è in funzione del target di clienti a cui essa è destinata. Infatti K-4Sales è utilizzata dagli agenti di vendita delle aziende che usufruiscono del servizio, i quali hanno bisogno di una rapida accessibilità ai dati e di una interfaccia semplice e veloce, che non faccia perdere tempo nel comporre un ordine, ma metta a disposizione tutte le funzionalità eseguendo i minor numero di passaggi.

5.2.1 Accesso al servizio

Le aziende che intendono utilizzare il servizio, devono contattare Kelyan S.p.a. per farsi creare un profilo. La registrazione infatti non è gestita dall'applicazione, ma viene affidata a chi controlla il *back-end*, che attiva gli utenti e i ruoli che l'azienda richiede, caricando la lista di clienti e prodotti relativi.
E' implementato il login attraverso 3 credenziali :

- username
- password
- codice azienda

L'utente che effettua il login può scegliere se "ricodare" l'accesso sul device in cui viene effettuato, per permettergli di evitare la procedura ad ogni accesso all'applicazione.

5.2.2 Utenti

Gli utenti che utilizzano questa app sono profili che lavorano per conto di una azienda che ha comprato il servizio. L'applicazione è pensata per l'utilizzo di due classi principali di utenti : agenti e amministratori.

L'utente "agente" può svolgere tutte le funzionalità di base :

- Aggiunta e modifica di un cliente
- Creazione / modifica / cancellazione di un ordine
- Visualizzazione informazioni d'accesso
- Gestione dispostivi connessi all'account
- Login/LogOut

L'utente "amministratore" può svolgere tutte le funzionalità di un agente, ad eccezione dell'aggiunta di un nuovo ordine, ma con alcune estensioni:

- aggiunta di un nuovo agente
- controllo dei device collegati ad un agente
- eliminazione di device collegati ad un agente
- eliminazione di un cliente
- modifica assegnazione di un cliente ad un agente.

Ogni agente è responsabile della gestione di uno o più clienti. I clienti sono associati ad un solo agente. Alla creazione di un nuovo cliente, se l'accesso è stato effettuato come agente, il nuovo cliente sarà associato ad esso, mentre se ad accedere è stato un amministratore, verrà proposta la lista di agenti che possono essergli associati.
Come indicato negli obiettivi riguardanti la felssibilità, non sempre gli utenti di questa applicazione trovano una personale applicabilità in queste due tipologie di accesso, ma talvolta richiedono che qualche azione o informazione sia bloccata o attivata per un certo utente. Per questo ho introdotto la gestione dei ruoli in maniera esplicita, dal *back-end* nel Cloud. E' possibile quindi creare nuovi ruoli con nuove funzionalità per soddisfare le esigenze dei clienti.

5.2.3 Aziende, clienti e prodotti

Il sistema può gestire l'utilizzo dell'app da parte di aziende diverse, ognuna identificata da un codice leggibile.
Ogni azienda che utilizza l'applicazione, possiede la sua lista clienti, ognuno caratterizzato dalle sue informazioni di base : ragione sociale, partita iva, indirzzo, telefono ecc..
Oltre ai clienti, sono gestiti tutti i prodotti, che possono essere diversi, in funzione dell'azienda. Per gestire questo ho implementato una apposita struttura dati che permette di catalogare prodotti di diversa natura, con attributi personalizzati (colore, dimensioni, taglie ecc..).

5.2.4 Ordini

Ogni utente può creare un ordine per ogni cliente che controlla. Ogni ordine è caricato con diversi prodotti attraverso l'interfaccia specializzata. Gli ordini possiedono inoltre una data di consegna e la possibilità di essere salvati in due modalità :

- sospendi ordine : l'ordine sospeso verrà salvato nello storico come tutti gli altri, ma sarà ancora possibile modificarlo, o eliminarlo
- chiudi ordine : l'ordine chiuso non è più modificabile e si attende che sia smaltito una volta scaduta la data di consegna.

Uno storico degli ordini è visualizzato in una sezione apposita, in cui viene effettuato un raggruppamento in base allo stato dell'ordine.

5.2.5 Configurazione

La sezione di configurazione fornisce le principali informazioni sulla sessione corrente dell'applicazione e permette, se si possiedono i permessi, di creare nuovi agenti. Inoltre è disponibile una lista di device connessi per effettuare il monitoraggio e l'eventuale cancellazione dei dispositivi connessi all'account, ovvero quelli per cui il login è stato effettuato ricordando l'accesso.
Un amministratore può visualizzare ed eliminare la lista di tutti i device dei suoi agenti, andando così a negare l'accesso automatico ad un profilo in caso di problemi.

5.3 Design della struttura dati

Il database per la gestione dei dati utili all'applicazione è stato interamente ridisegnato all'interno del framework IDC per sfruttare e avere prova di tutti i vantaggi che questa soluzione porta, rispetto all'importazione del database già esistente per l'applicazione originale.
Come spiegato nella sezione riguardante le specifiche di IDC, è comunque sempre possibile collegare un database esterno, qualora fossero necessari alcuni dati accessibili solamente da database aziendali privati.

5.3.1 Schema Entità-Relazioni

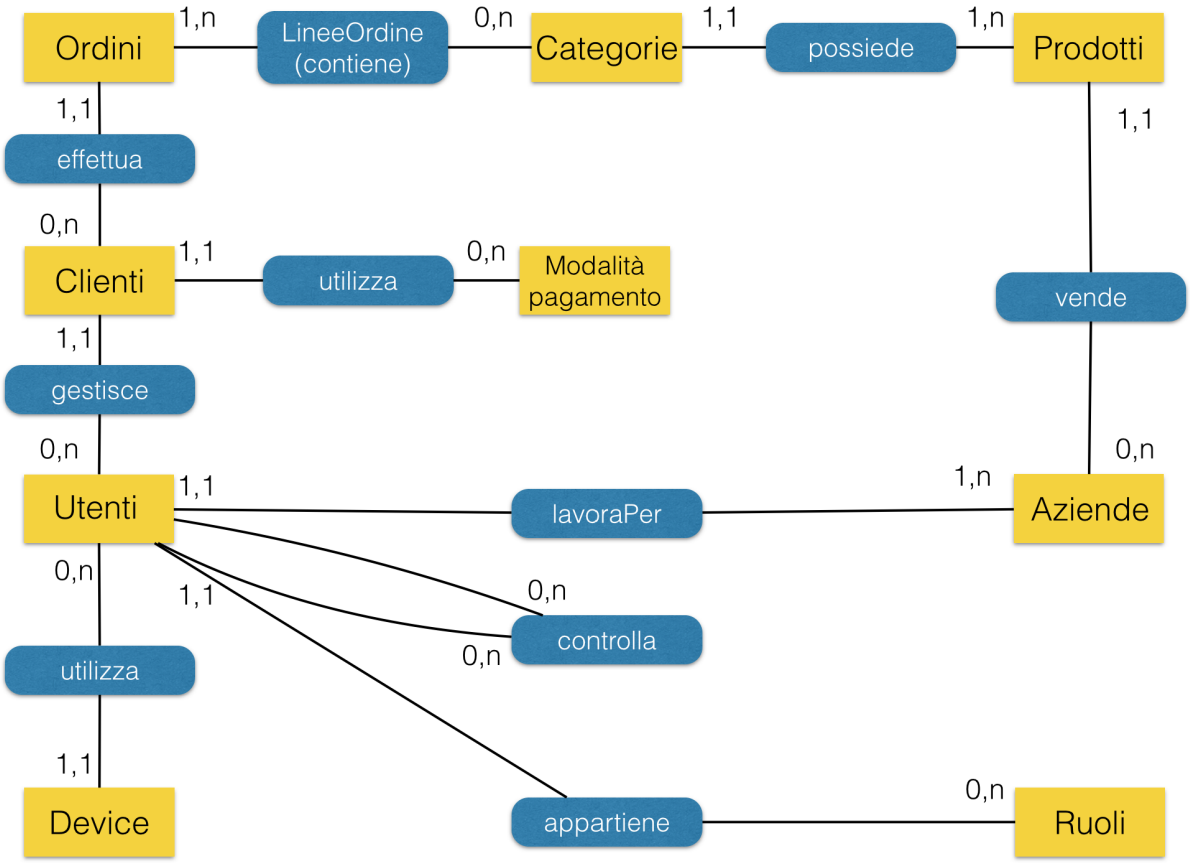

Figura 5.1: Schema entità relazioni del database

5.3.2 Analisi delle relazioni

Ogni tabella possiede una chiave primaria chiamata "ID"+"nomeTabella". Il campo è di tipo UUID (*Universally Unique Identifier*), consigliato per i campi identificativi utilizzati come chiave primaria. Esso costituisce una stringa di 24 caratteri generati in automatico dal sistema. L'utilizzo di questo tipo è fortemente consigliato dal framework in quanto permette una maggior facilità nella gestione della sincronizzazione fra database nel Cloud e quelli nei dispositivi, evitando problemi di conflitto. Inoltre il tipo UUID è auto-increment, e permette di non preoccuparsene in fase di sviluppo perché ogni nuovo record aggiunto sarà automaticamente associato ad una chiave univoca.

- **Aziende**: questa tabella contiene i record corrispondenti alle aziende iscritte al servizio K-4Sales. Oltre alla chiave primaria UUID, esse possiedono un codice leggibile fornito in fase di registrazione, che è uno dei parametri richiesti per effettuare il login.

- **Utenti**: sono salvati qui tutti gli utenti delle aziende, siano essi agenti, amministratori, o di altri ruoli. Ogni utente fa parte di una sola azienda ed appartiene ad un solo ruolo, quindi sono presenti due chiavi secondarie che rappresentano le due relazioni 1-n. Sono salvati qui gli hash delle password di ogni utente per garantire riservatezza su di esse, evitando di salvarle in chiaro.

- **Controlla**: rappresenta una relazione ricorsiva. Ogni utente può controllare un altro utente. Ad esempio, un amministratore può controllare tutti gli agenti della sua azienda. Non è esclusa però la possibilità dell'inserimento di altri ruoli, come ad esempio il "capoarea", che ipoteticamente potrà controllare tutti gli agenti, ma sarà controllato a sua volta dall'amministratore. Ho inserito questa relazione per gestire la visualizzazione dei dati contestuali per gli accessi con profili che non sono strettamente collegati ai clienti.
  Ad esempio, un amministratore non gestisce nessun cliente direttamente, ma grazie alla relazione "controlla", sarà possibile andare a recuperare i clienti gestiti da tutti gli utenti da lui controllati.

- **Ruoli**: questa tabella è responsabile della gestione dei permessi associati ad un ruolo. Sono caricati principalmente due ruoli, agente e

amministratore, ma possono essere aggiunti altri, a seconda delle richieste dell'azienda cliente.
Per aggiungere un nuovo ruolo è sufficiente indicarne il nome e la descrizione, e attivare o meno i flag che gestiscono le azioni che esso può svolgere. Ad esempio impostando a *false* "aggiungiClienti", verrà negata la possibilità di aggiungere un nuovo cliente da parte del ruolo associato.
Molto importante è il campo "gestoreCliente". Esso permette di capire, in fase di caricamento dei dati, se il ruolo è responsabile della gestione diretta di un cliente, oppure è un ruolo per gestire altri utenti, come l'amministratore o un ipotetico capoarea. I flag per gestire i permessi possono essere estesi, andando a attivare o meno diverse azioni all'interno dell'applicazione.

- **Clienti**: Ogni cliente è caratterizzato da una serie di attributi che forniscono le informazioni principali per la gestione di un ordine, come "Ragione sociale", "Partita IVA" o "Indirizzo". Ogni cliente è gestito da un solo utente, ed è associato ad un solo metodo di pagamento. Infatti anche la tabella clienti contiene due chiavi secondarie per le due relazioni 1-n.

- **Device**: Tabella responsabile della gestione dei device. Ogni record rappresenta un device fisico collegato ad un solo cliente. Il device è caratterizzato dal nome, dal codice univoco del dispositivo e dall'informazione temporale dell'ultimo accesso.

- **Ordini**: Ogni cliente può essere associato a più ordini, ma un ordine appartiene ad un solo cliente. I record della tabella ordine indicano i dati generali di un ordine, come la data in cui è stato effettuato, la data di scadenza, l'importo totale e lo stato dell'ordine. In questa tabella non vengono salvate le informazioni relative ai prodotti caricati in ogni ordine.

- **Prodotti**: Questa tabella contiene i prodotti venduti da ogni azienda. In particolare essa contiene solamente le informazioni generali di un prodotto, come il prezzo e il nome, oltre alla chiave esterna per collegarlo all'azienda relativa.

- **Categorie**: ho creato questa tabella che rappresenta le specializzazioni dei prodotti per riuscire a gestire le differenze di attributi dei prodotti venduti dalla varie aziende.
  Ogni azienda, vendendo prodotti diversi, ha bisogno di salvare per essi attributi che possono essere di natura diversa. Ad esempio un'azienda di abbigliamento, sarà interessata a colore e alla taglia dei suoi abiti, mentre un'azienda dolciaria, vorrà specificare le calorie e la data di scadenza dei suoi biscotti.
  La tabella Categorie gestisce questo problema, evitando di creare una nuova relazione per ogni tipologia di azienda, rendendo lo schema del database flessibile per ogni tipo di cliente.
  Per fare questo ho inserito in Categorie tre campi fondamentali :

  1. nome: contiene il nome dell'attributo da associare al prodotto, come "taglia" oppure "colore".
  2. valore: per ogni nome di attributo, contiene i valori associati ad esso. Ad esempio per il nome "taglia", potranno esserci tre record, che contengono rispettivamente i valori "L", "M", "S" se il prodotto in questione è disponibile in queste tre taglie.
  3. disponibile : campo booleano che indica la disponibilità del prodotto in questa categoria. Se a false, non sarà possibile scegliere questo record in fase di creazione di un ordine.

  Ogni prodotto può essere associato a più categorie se ne possiede le caratteristiche. Se un prodotto non possiede la categoria associata ad un determinato nome, viene comunque inserita una tupla con "valore = *null*".
  In seguito è presentato un esempio per gestire il colore di alcuni prodotti nell'applicazione sviluppata. Si può notare come il prodotto con identificativo "uCsp. . . " compare in diverse tipologie di colore, mentre per il prodotto nell'ultima riga "RZ/uy. . . " non è prevista una scelta di colore. Il colore "Arancione" inoltre non è disponibile nella configurazione attuale.
  Questo sistema mi permette di essere flessibile sulla tipologia di attributi associati ad ogni prodotto e qualora un azienda decida di aggiungere o eliminare una categoria per uno dei suoi prodotti, è necessario agire su questa tabella, e la modifica sarà riflessa sulla creazione degli

ordini futuri. L'attributo disponibile permette di disattivare una categoria, che non viene eliminata dal database perché è utile a comporre le copie degli ordini che l'hanno seleziona in passato.

#	IDCATEGORIA	NOME	VALORE	DISPONIBILE	IDPRODOTTO
1	Dv4kQRgT+EVnFyX7GEpXRA==	colore	Rosso	true	uCspFbUZG6JYxJTvAVVtMg==
2	kks2Hovu56XdmoaBoF9Xlg==	colore	Blu	true	uCspFbUZG6JYxJTvAVVtMg==
3	4KXGhu8nICHdHhWr6ay3gg==	colore	Azzurro chiaro	true	uCspFbUZG6JYxJTvAVVtMg==
4	AHSAaDhzvGBbjS/5ebbsDg==	colore	Arancione	false	uCspFbUZG6JYxJTvAVVtMg==
5	IUP0RZ5jVHkhi33Cex9r9w==	colore	grigio siderale	true	uCspFbUZG6JYxJTvAVVtMg==
6	0sNXmnePwHnojUFJZiAuqA==	colore	rosso fuoco	true	uCspFbUZG6JYxJTvAVVtMg==
7	9YCypvUuOqKGg8oMArgM9w==	colore	null	true	RZ/uyBd7L9KcXgjoBhtnmw==

Figura 5.2: menu IonTab per navigare nelle pagine

- **LineeOrdine**: rappresenta la relazione n-n fra Ordini e Categorie. La composizione di un ordine avviene infatti attraverso le categorie specifiche dei prodotti e non sui prodotti stessi. Ad un ordine quindi è possibile collegare più categorie appartenenti allo stesso prodotto.
  La tabella ha una chiave primaria singola "IDLineeOrdine", per permettere ad un ordine di essere in relazione con più categorie e alle singole categorie di essere collegate a più ordini.

5.4 Pagine e layout

In questa sezione vengono illustrate le videate che compongono l'applicazione, indicandone i principali componenti.

5.4.1 Navigazione e composizione delle pagine

L'applicazione si divide in tre sezioni principali, oltre alla schermata di login:

- Clienti
- Storico
- Configurazione

La navigazione attraverso le videate è gestita da un componente del framework Ionic, chiamato *IonTabs*. Esso contiene tre *IonTab*, che a loro volta contengono gli oggetti delle videate che vogliono mostrare. Infatti,

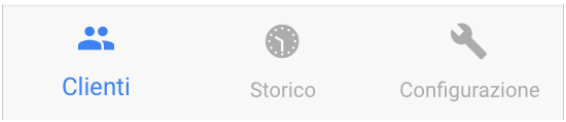

Figura 5.3: menu IonTab per navigare nelle pagine

ogni videata creata, può essere intesa come un oggetto di layout se ad essa viene settata la proprietà *design time* e quindi aggiunta all'interfaccia di altre pagine. In questo caso le tre pagine principali sono attive in "design time", e vengono aggiunte ai tre oggetti dello *IonTabs*. La pagina che contiene lo IonTabs è caricata in seguito al login e permette di mantenere la struttura del menu in basso fissa, al variare delle diverse sezioni.
Ogni pagina è composta grazie ad una classe preimpostata del framework Ionic : *IonPage*. Questa classe, attraverso l'oggetto grafico *IonStdPage*, predispone una videata ad essere composta rispettando i canoni generali della UI di una applicazione comune. La classe è molto utile per gestire qualsiasi tipo di device, in quanto adatterà le dimensioni e la posizione delle icone in funzione dello schermo su cui viene istanziata.

Essa divide la pagina in tre sezioni:

- Header : contiene l'oggetto *Navbar*, il quale gestisce il titolo della schermata e il "BackButton" per scorrere indietro nelle videate.
- Content : è corpo della pagina, in cui si inseriscono gli elementi principali
- Footer : sezione predisposta all'inserimento dei pulsanti di modifica.

5.4.2 Pagina di login

La videata permette di inserire le credenziali e selezionare il campo "Ricordami" per evitare di effettuare il login al prossimo accesso. Una funzione scritta nel *back-end* consentirà l'accesso in caso di dati corretti.

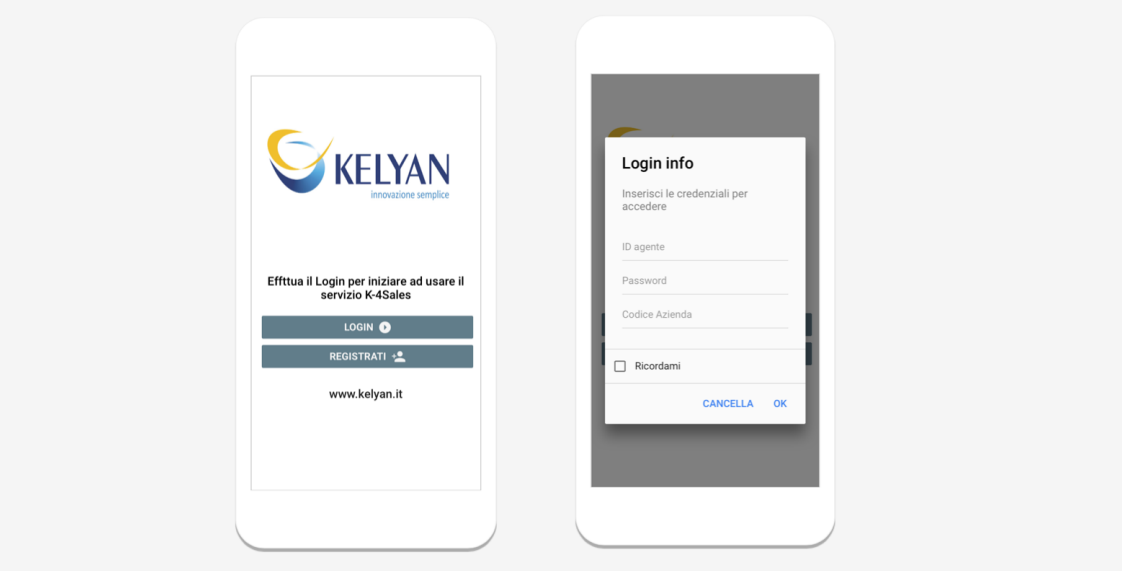

Figura 5.4: Schermate di login

5.4.3 Gestione clienti

Questa è la prima View che viene mostrata all'accesso. In primo piano troviamo la lista di clienti a cui l'utente della sessione attuale è collegato. Attraverso la segmentBar, posta in cima alla lista, è possibile, visualizzare la barra di ricerca, e se si hanno i permessi, la lista di agenti controllati e i clienti sotto la gestione di ogni agente.
Cliccando su "+nuovo", se disponibile, si accede alla sezione dedicata all'inserimento di un nuovo cliente, nella quale sono richiesti i campi, obbligatori e non, per poi effettuare il salvataggio.
Un campo significativo è "agente" : se l'utente che ha effettuato l'accesso è un gestore diretto di clienti, questo campo non viene mostrato perché è automaticamente completato con il proprio ID univoco. Al contrario, se chi aggiunge il cliente non è un gestore diretto, come ad esempio un amministratore, è possibile scegliere un utente controllato a cui affidare il cliente.
Tornando alla lista principale, cliccando su un qualsiasi elemento, si apre il dettaglio del cliente, che mostra informazioni aggiuntive rispetto a quelle nella lista. In questa sezione è possibile modificare, eliminare o aggiungere un ordine per il cliente in base al ruolo di appartenenza. Per comporre la

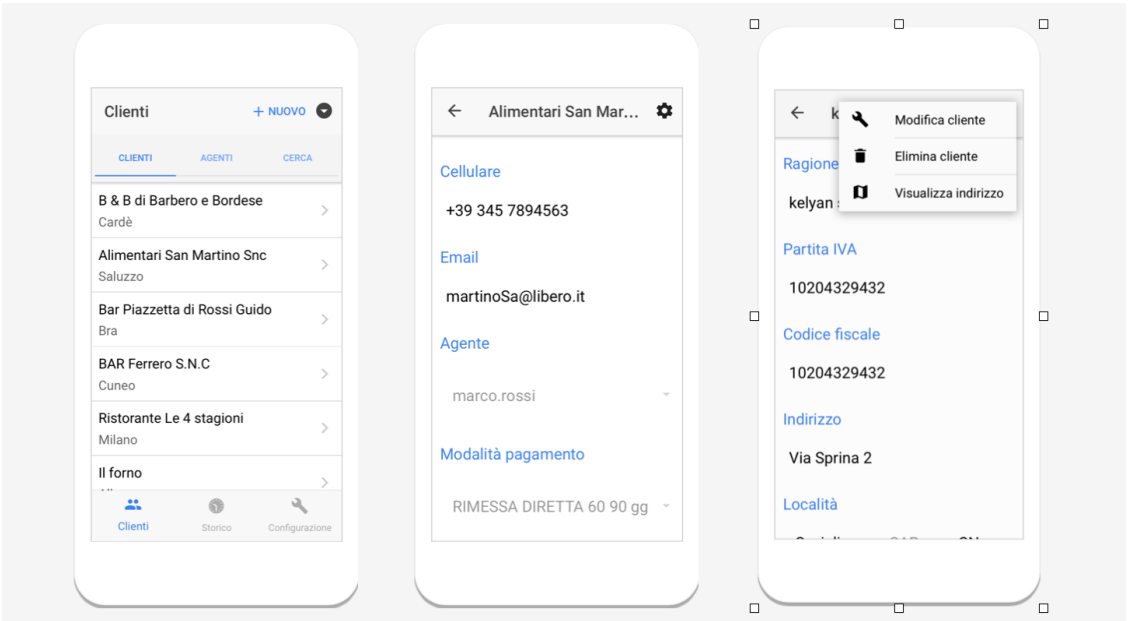

Figura 5.5: Schermate per la gestione dei clienti

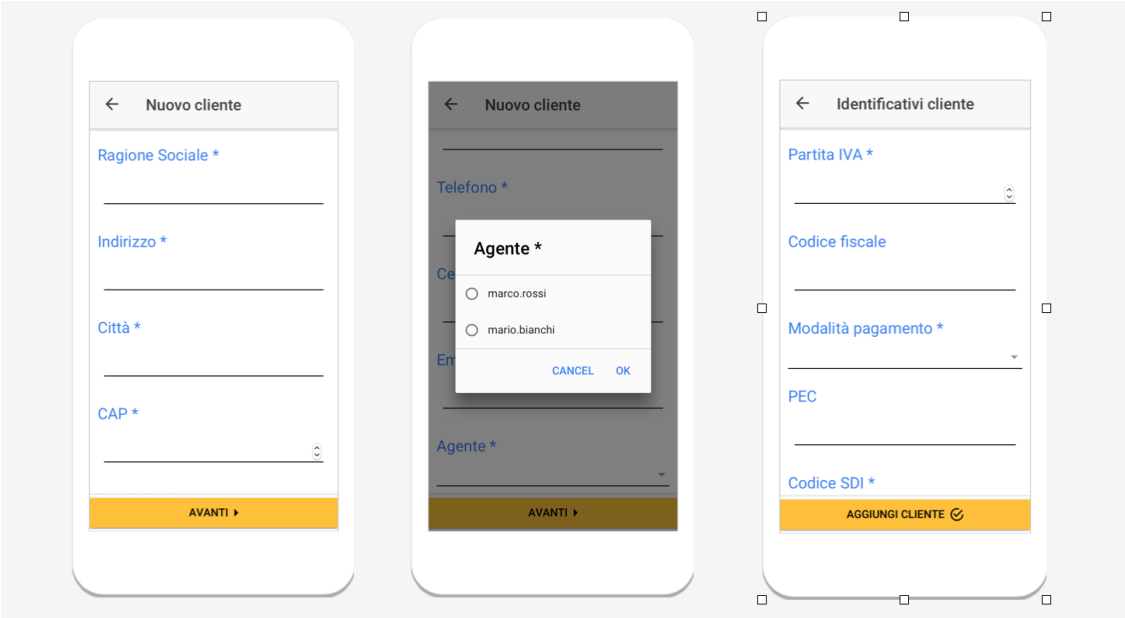

Figura 5.6: Schermate per l'aggiunta di nuovi clienti

videata di dettaglio, ho utilizzato un elemento molto importante del framework Ionic: *IonGrid*. Esso permette di organizzare la videata mediante una griglia responsive.
La schermata del dettaglio cliente è un esempio calzante per l'applicazione della IonGrid. Infatti, su smartphone, è comodo visualizzare le informazioni dei clienti in una lista verticale, mentre su tablet risulterebbe una soluzione poco funzionale, che non sfrutterebbe lo spazio in più a disposizione nella videata. Per questo *IonGrid*, permette di mostrare per ogni riga un numero di elementi adeguato alle dimensioni dello schermo in uso. Ogni riga della griglia è divisa in 12 parti e per ogni colonna aggiunta ad una riga, è possibile specificare quante di queste parti dovrà occupare, in base al device e all'orientazione dello schermo. Per ogni cliente è infine possibile visualizza la mappa relativa all'indirizzo, cliccando sul apposita voce del popup menu.

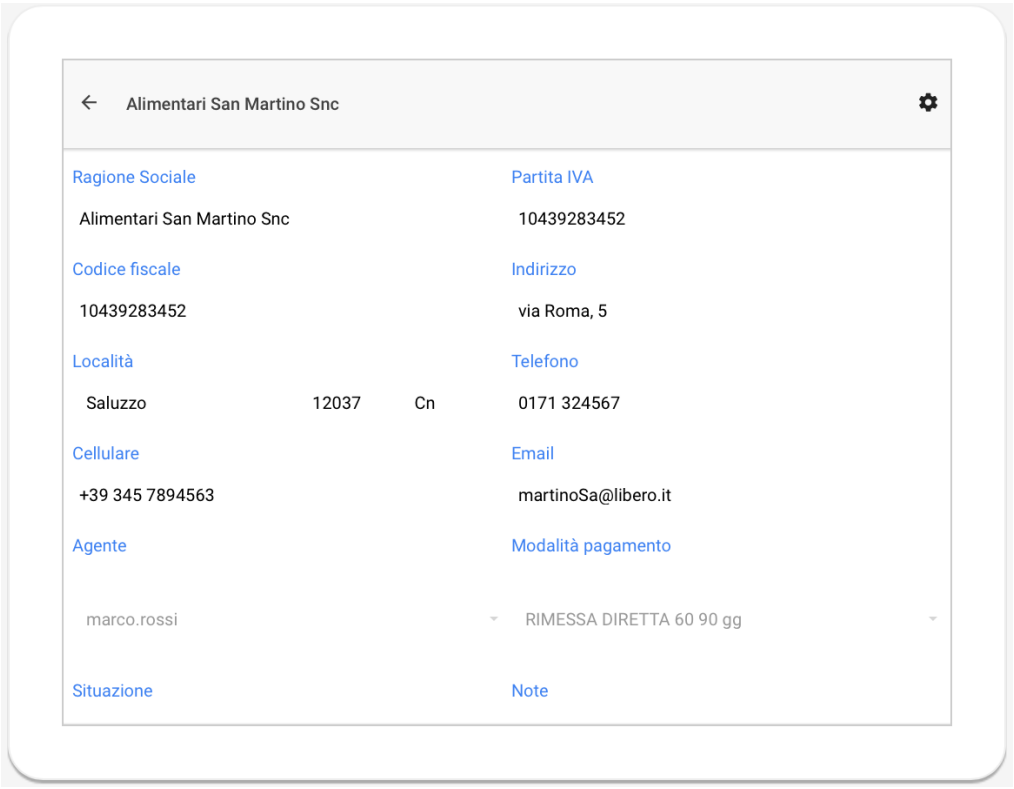

Figura 5.7: Visualizzazione dettaglio cliente su Tablet orizzontale, adattato allo schermo orizzontale.

5.4.4 Ordini

La schermata degli ordini permette di caricare i prodotti con i loro attributi, visualizzare la somma del prezzo totale e inserire la data di consegna. Per poter visualizzare in maniera ordinata le linee d'ordine, ho creato due diverse view, una per smartphone e una per tablet o desktop. Quando l'app viene utilizzata su tablet, viene mostrata una tabella che occupa in maniera ottimizzata il maggior spazio a disposizione. Su smartphone le linee d'ordine sono inserite all'interno di elementi chiamati *IonCard*, che funzionano da contenitori. In entrambi i casi è possibile modificare le linee o eliminarle, cliccando sul prodotto desiderato. L'aggiunta di un ordine

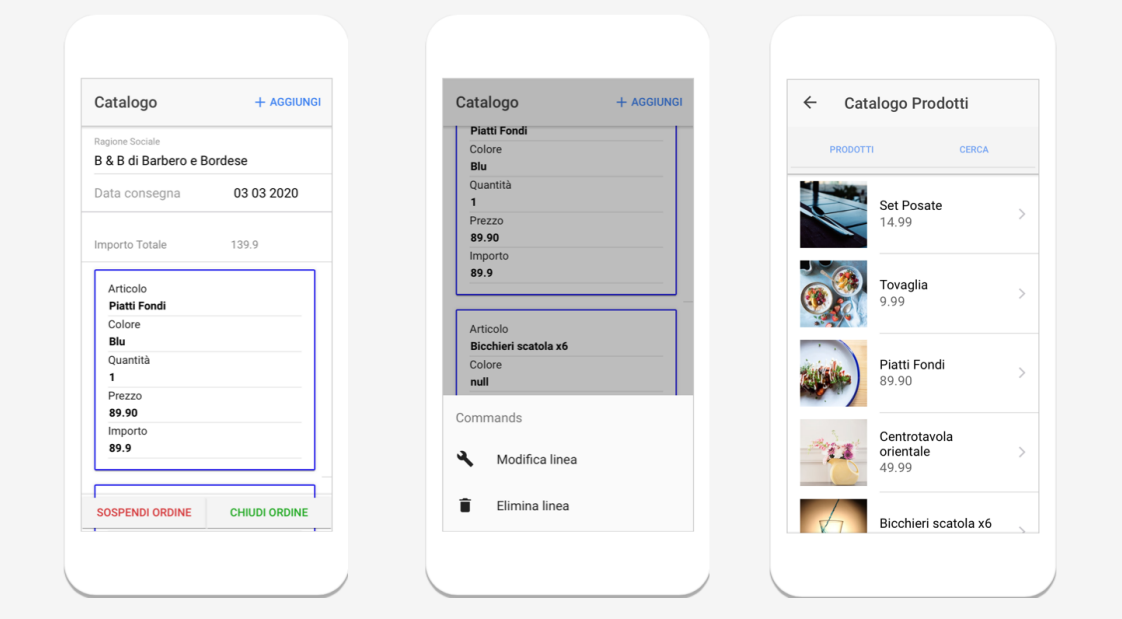

Figura 5.8: Schermate per la composizione di un ordine

porta ad altre due schermate; la prima permette di scegliere il prodotto da catalogo dell'azienda relativa, la seconda visualizza il layout per la selezione degli attributi relativi al prodotto selezionato.
Infine è possibile chiudere l'ordine, che verrà inviato una volta raggiunta la data di consegna, oppure sospenderlo per poterlo modificare in futuro.

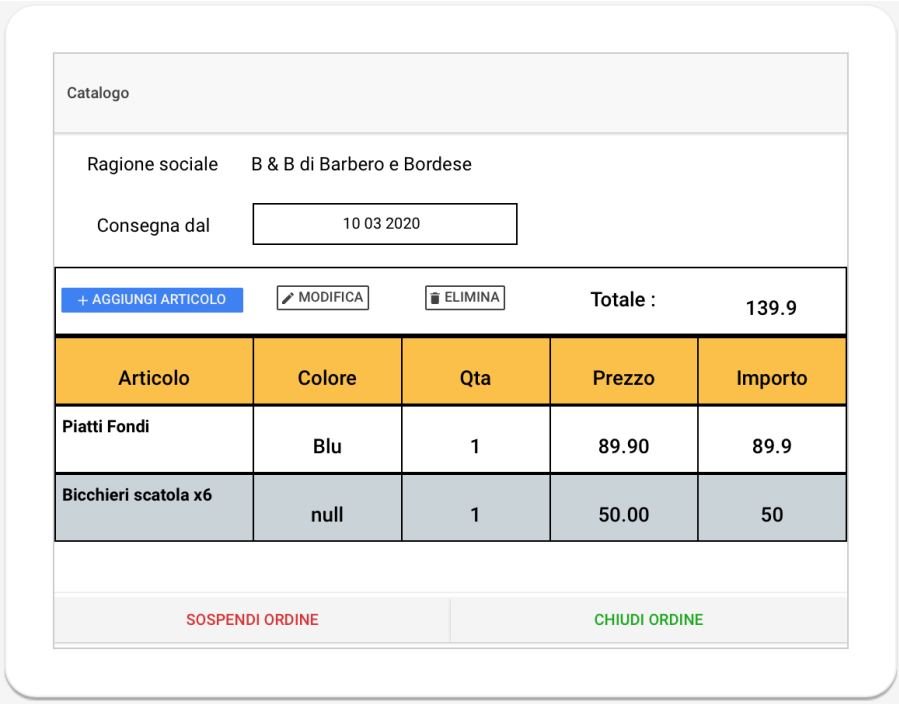

Figura 5.9: Visualizzazione della tabella ordini su tablet

Figura 5.10: Scelta delle categorie del prodotto su tablet

5.4.5 Storico

Storico è la seconda sezione navigabile attraverso il Tab menu. Si possono trovare qua tutti gli ordini effettuati, se diretti gestori di un cliente, o tutti gli ordini subordinati agli utenti che il ruolo controlla.
Gli ordini sono divisi in tre categorie attraverso una toolbar posizionata nell'header della pagina :

- da inviare : ordini chiusi la cui data di consegna non ha superato la data odierna.
- inviati : ordini chiusi la cui data di consegna ha superato o è il giorno stesso.
- sospesi : ordini sospesi, qualsiasi data

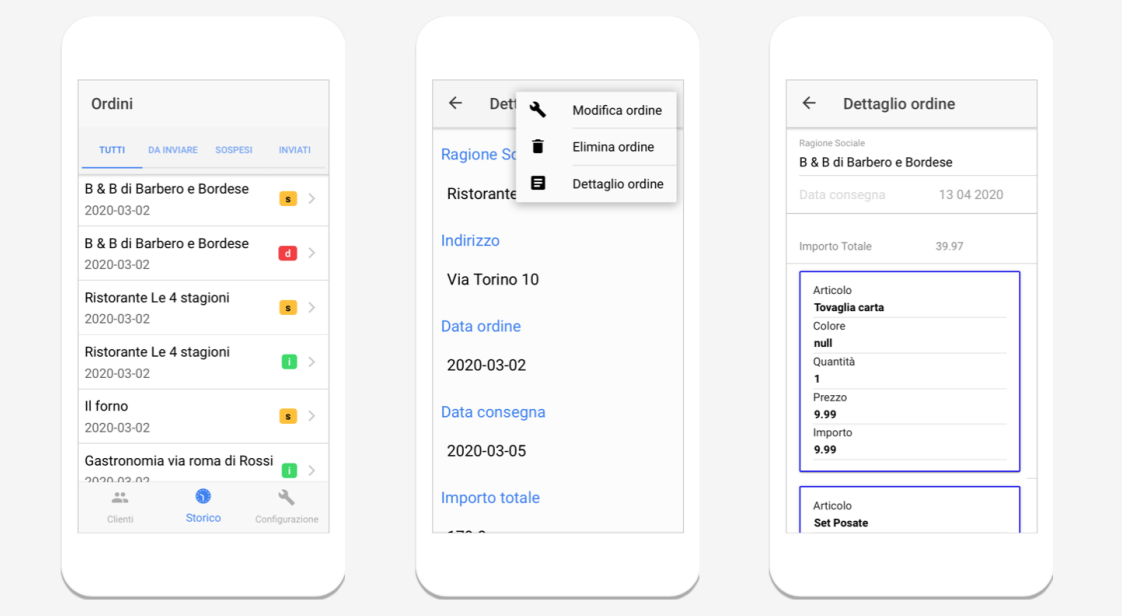

Figura 5.11: Visualizzazione delle videate della sezione "storico"

Ad ogni ordine è associato un tag sulla riga per classificarlo velocemente. Selezionando un ordine è possibile visualizzare le informazioni generali, come l'importo totale o la data di consegna, ma non i prodotti inseriti.
Per fare questo è necessario aprire il dettaglio ordine dal popup menu, che richiama la schermata per la composizione dell'ordine, ma in sola lettura,

mostrando la tabella associata. Soltanto per gli ordini sospesi è possibile modificare, aggiungere o eliminare le linee d'ordine, e cambiare lo stato, da sospeso a chiuso.

5.4.6 Configurazione

L'ultima sezione è quella dedicata alla configurazione. Nella prima parte della schermata vengono visualizzate le informazioni principali relative alla sessione attuale, mentre nella sezione configurazione sono disponibili alcune azioni in base al ruolo dell'utente :

Aggiungi nuovo utente: solamente un amministratore può aggiungere nuovi utenti, ai quali è possibile assegnare un username, una password e un ruolo fra quelli già disponibili nel sistema. Il nuovo utente inserito sarà controllato dall'amministratore che lo crea.

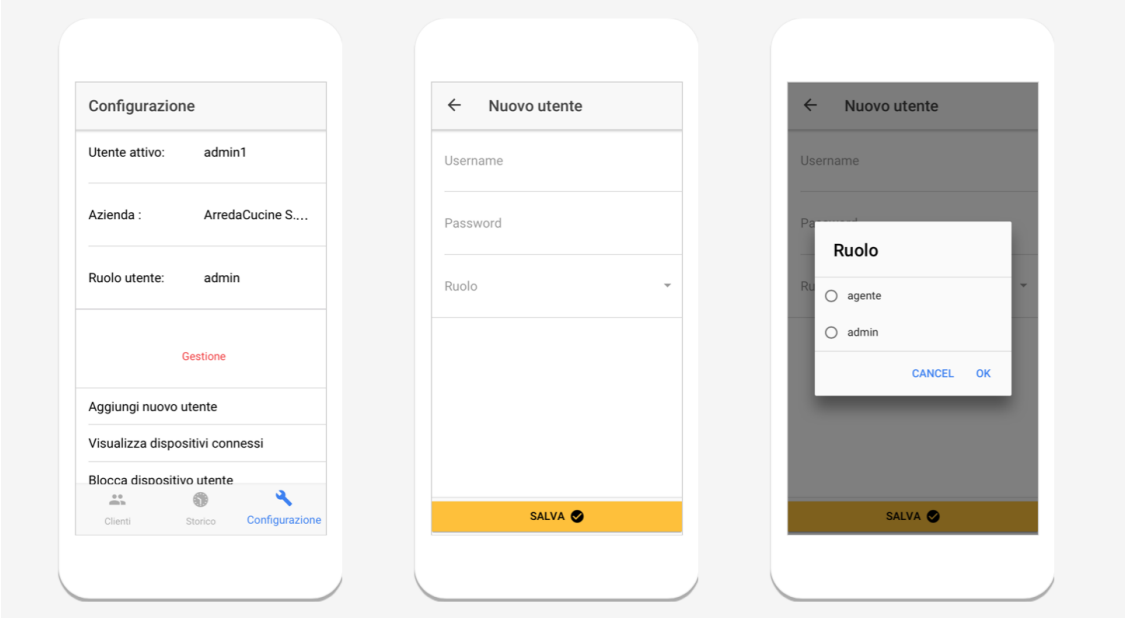

Figura 5.12: Informazioni relative alla configurazione e aggiunta di un nuovo profilo

Visualizza dispositivi connessi: permette di visualizzare tutti i dispositivi per i quali si è effettuato il login con l'opzione "Ricordami", con la data e l'ora dell'ultimo accesso. Attraverso l'oggettoo *IonSwipe*, ho inserito l'azione di scorrimento sulla riga di una lista, ormai presente su tutti i sistemi operativi da qualche anno. Trascinando verso sinistra una riga relativa ad un device è possibile eliminarlo dai dispositivi connessi all'account. Se il dispositivo è quello in uso, verrà effettuato anche il logout.

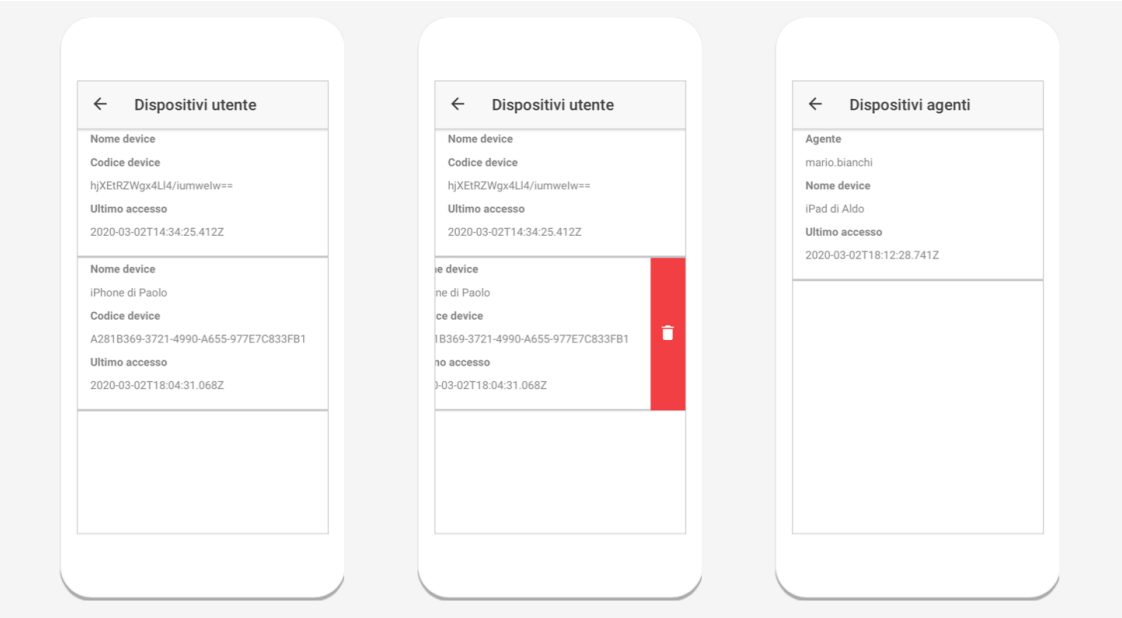

Figura 5.13: Lista dei device connessi con possibilità di rimozione del dispisitivo

Blocca dispositivo utente: Per l'amministratore è disponibile la possibilità di eliminare anche un dispositivo relativo ad un utente da lui controllato. Questa opzione permette di scollegare un dispositivo che è stato smarrito o viene sostituito, in modo da evitare accessi indesiderati e un possibile danneggiamento dei dati di vendita.

5.5 Tecniche e algoritmi

5.5.1 Classe Pages

Una classe molto utile alla navigazione all'interno dell'applicazione è Pages. Essa estende la classe *MainPage* del framework Ionic ed è possibile gestire con facilità il cambio di schermata in funzione degli eventi sull'applicazione.
Attraverso i suoi due metodi principali, push e pop, è possibile gestire le chiamate alle videate come una struttura stack.

- **Metodo push**: il metodo push viene chiamato quando si intende cambiare videata "in avanti", seguendo il flusso di navigazione. Il metododo permette di specificare alcuni parametri di default e altri personalizzati.
  Il primo parametro è sempre l'oggetto app, il secondo è la classe appartenente alla videata che si vuole mostrare, e il terzo , racchiuso tra parentesi graffe, contiene la lista dei parametri standard, più quelli personalizzati.
  Nel listato 5.1 riportato in seguito, la chiamata imposta due parametri standard. Il primo, "root", indica che la videata chiamata dovrà essere usata come base per lo stack di navigazione, mentre "remove" permette di rimuovere la pagina chiamante dallo stack.
  Nel listato 5.2 invece i parametri sono personalizzati, in base ai dati che si vogliono trasmettere alla classe della videata chiamata.

```
1     App.Pages.push(app, App.ViewTabs, {
2         root : true,
3         remove : true
4     })
```

Listato 5.1: Chiamata a metodo push parametri standard

```
1     App.Pages.push(app, App.DettaglioCliente, {
2         cliente : this.row.document,
3         filtro : view.filtro
4     })
```

Listato 5.2: Chiamata a metodo push parametri personalizzati

- **Metodo pop**: permette di ritornare indietro nelle videate seguendo lo stack di navigazione. Il metodo richiede come primo parametro l'oggetto app, e come secondo il numero di view da rimuovere.
  L'oggetto *IonStdPage* della classe IonPage analizzato nelle sezioni precedenti, gestisce automaticamente il "pop" al click sul *back button* se esso è attivo. Per questo il metodo pop è utilizzato quando la logica della videata richiede esplicitamente un ritorno al contesto precedente, come ad esempio un pulsante per il salvataggio.

```
1     App.Pages.pop(app, 2);
```

Listato 5.3: Chiamata a metodo pop.ggg

5.5.2 Back-end e Documenti

Ho inserito all'interno del back-end una libreria (4KSBE), responsabile della gestione dei documenti. Ogni tabella del database è associata ad un documento che contiene al suo interno le proprietà, dirette o derviate, e i metodi per gestire le azioni sui suoi dati. Per utilizzare i dati dei docuementi nel *front-end*, ho utilizzato principalmente le DataMap e le funzioni *loadByKey*, *loadCollection* e *save*.
Tutte le funzioni scritte nel *back-end* hanno sempre a disposizione l'oggetto app, che viene passato ad esse obbligatoriamente da qualsiasi chiamata per ottenere le proprietà relative alla sessione di utilizzo dell'applicazione

5.5.3 Login e Logout

LoginByDeviceID
La prima istruzione che viene effettuata nel metodo *onStart* è una chiamata alla funzione *loginByDeviceID* nel back-end. Essa si occupa di recuperare l'ID del device dall'oggetto *app.device*, e controllare, mediante una *loadByKey*, se il codice è associato ad uno dei device collegati agli utenti. La funzione ritorna un oggetto che, se diverso da null, permette di accedere immediatamente alla schermata clienti. Altrimenti viene mostrata la videata di login.

gestioneLogin
Se il device non è registrato, viene chiamata la funzione *getionLogin*, la

quale effettua un controllo sulla corrispondenza delle credenziali inserite con un utente nel database. In caso di credenziali errate, viene ritornato false. Al contrario se le credenziali sono corrette, e viene attivato il flag "ricordami", vengono lette le informazioni del dispositivo dall'oggetto *app.device* e inserite in un nuovo documento che viene salvato, e quindi aggiunto al database.

gestioneLogout
Questa funzione libera le proprietà di sessione relative all'utente che ha effettuato l'accesso e rimuove il dispositivo ricordato, se presente, collegato all'account.

5.5.4 Inizializzazione delle proprietà

E' possibile associare all'intera applicazione metodi, classi e proprietà globali, che possono essere utilizzate grazie all'oggetto *app* in qualsiasi contesto del *front-end* e del *back-end*. Per questo ho ritenuto utile creare 3 proprietà di applicazione per poter accedere comodamente, senza bisogno di ulteriori query o caricamenti di documenti, alle informazioni di sessione. Esse sono caricate in fase di login, sia per device che per credenziali, e liberate nel logout:

- **account**: proprietà di tipo "Utenti" che contiene l'istanza del documento dell'utente che ha effettuato l'accesso
- **ruolo**: proprietà utile per gestire i permessi delle azioni che sono legati al ruolo
- **deviceOn**: proprietà che contiene i dati legati al device "rircordato", come il codice univoco o l'ultimo accesso.

```
1 app.account = agent;
2 app.ruolo = yield App.K4SBE.Ruoli.loadByKey(app,
    app.account.IDRuolo);
3 app.deviceOn = device;
```

Listato 5.4: Inizializzazione delle proprietà di applicazione

5.5.5 Metodo OnLoad e passaggio parametri

Il metodo *OnLoad* è notificato all'apertura delle videate. Esso è uno dei metodi più utilizzati quando si sviluppa all'interno del framework IDC, in quanto permette di completare la configurazione della videata, mostrando o nascondendo elementi, caricare le proprietà di videata e gestire i dati che serviranno alle DataMap.
All'interno di questo metodo ho gestito l'assegnazione dei permessi in relazione alla proprietà globale "ruolo" caricata in fase di login. Principalmente ho agito sulle proprietà degli oggetti da nascondere e, in alcuni casi, come nei menu di popup, ho gestito dei vettori di pulsanti caricati secondo le azioni permesse.
Il codice 5.5 in seguito mostra la creazione del menu di popup del dettaglio di un cliente controllando se l'utente è autorizzato o meno a effettuare le diverse azioni.

```
1 var menuitems = [];
2 if (app.ruolo.modificaClienti)
3     menuitems.push({id : 1, icon : "build", title :
        "Modifica␣cliente"});
4 if (app.ruolo.eliminaClienti)
5     menuitems.push({id : 2, icon : "trash", title :
        "Elimina␣cliente"});
6 if (app.ruolo.gestoreCliente)
7     menuitems.push({id : 3, icon : "list-box", title :
        "Aggiungi␣ordine"});
8 menuitems.push({id : 4, icon : "map", title :
    "Visualizza␣indirizzo"});
9 var b = yield app.popup({type : "menu", refObj :
    $menuBtn.id, items : menuitems});
```

Listato 5.5: Composizione del menu popup di DettaglioCliente

Il metodo onLoad è anche utilizzato per gestire i parametri che riceve la videata aperta da quella chiamante, attraverso il metodo push della classe Pages.
Generalmente ogni videata dispone dei dati utili per gestire il suo contesto, che solitamente sono le DataMap e le proprietà. Essi vanno inizializzati in base ai parametri ricevuti.

Dall'esempio 5.2 si può notare come ho gestito una chiamata ricorrente all'iterno dell'applicazione. Un elemento di una lista viene selezionato e di conseguenza deve essere aperta una seconda videata, che si popola dei dati dell'elemento appena premuto. La chiamata infatti è effettuata sul click di un elemento della lista di clienti della pagina principale. Ad essa sono associati due parametri personalizzati :

- cliente : contiene l'istanza del cliente relativa alla datamap associata alla lista. E' ottenibile grazie all'istruzione "*this.row.document*", che ottiene dalla riga selezionata (this), il documento relativo.

- filtro : variabile della videata "clienti" utile anche al dettaglio.

Sull'evento *onLoad* della *View* "DettaglioCliente" in 5.6, grazie al parametro options della funzione, recupero i due parametri passati dalla chiamata e li assegno alle variabili interessate, in questo caso la proprietà "istanzaCliente" e la DataMap "clienteDM".

```
1 view.istanzaCliente = options.cliente;
2 $clienteDM.document = view.istanzaCliente;
3 $title.innerText = view.istanzaCliente.ragioneSociale;
```

Listato 5.6: Istruzioni di assegnazione nell' evento onLoad di DettaglioCliente

5.5.6 Caricamento DataMap e Filtri

Come già discusso relativamente agli strumenti di IDC, ho utilizzato con frequenza le DataMap per sfruttare la loro velocità nella gestione dei dati rispetto all'interfaccia con l'utente. Praticamente ogni videata contiene uno o più Datamap, che sono caricate spesso nei metodi *onLoad*.
Di seguito riporto un esempio della gestione della DataMap associata alla sezione dei clienti, alla quale è stato necessario applicare un filtro in maniera particolare . Prima di tutto l' onLoad della *View* "Utenti", utilizza una funzione nel *back-end* per impostare il filtro da applicare alla DataMap.
La funzione è "filterIDUtenti" e lavora in due modi diversi a seconda del ruolo attivo :

1. Se il ruolo è un “gestoreCliente”, allora funzione ritorna semplicemente l’ “IDUtente”.

2. Al contrario, la funzione crea una stringa di IDUtenti che sono sotto il controllo del ruolo attivo, eseguendo una query strutturata in maniera esplicita sulla relazione “Controlla” del database.

Il risultato è salvato nella proprietà “filtro”, che verrà utilizzata per ottenere la lista degli agenti sotto il controllo dell’utente attivo.
A questo punto risulta molto semplice selezionare il clienti corretti, utilizzando la funzione *addFilter* sulla datamap, passando come parametro la lista di IDUtente accettati contenuti nel filtro appena creato.

```
1 view.filtro = yield
     App.K4SBE.Utenti.filterIDUtenti(app);
2 $idClientiDM.addFilter("IDUtente", view.filtro);
3 yield $idClientiDM.load();
```

Listato 5.7: Utilizzo filtro su un parametro di una DataMap

E’ possibile inserire numerose tipologie di filtri in funzione della selezione che si vuole ottenere. In seguito un esempio relativo alla DataMap per lo storico, in cui vengono filtrati gli ordini in base alla data di consegna.

```
1 var oggi = app.locale.today();
2 $OrdiniStoricoDM.clearFilters(App.K4SBE.Ordini.statoOrdine);
3 $OrdiniStoricoDM.clearFilters(App.K4SBE.Ordini.dataConsegna);
4 $OrdiniStoricoDM.addFilter("statoOrdine", "chiuso");
5 $OrdiniStoricoDM.addFilter("dataConsegna", ">" +
     oggi.format("YYYY-MM-DD"));
6 yield $OrdiniStoricoDM.reload();
```

Listato 5.8: Utilizzo filtro sulle date

5.5.7 Caricamento icone prodotti

Nella sezione relativa agli ordini, quando si sceglie un prodotto vengono caricate delle icone vicino alla descrizione e al prezzo. Le immagini caricate non sono salvate nel database locale ma ricevute grazie ad una API fornita da unsplash.com, che permette di ottenere immagini di varie dimensioni. Le immagini provengono da un sito di prova, ma potrebbero essere ottenute da un Web service privato interno all'azienda cliente che fornisce i dati dei suoi prodotti.
Ho introdotto questa funzione per mostrare la semplicità dell'implementazione di dati esterni, che non per forza devono essere sempre all'interno del database del cloud IDC.

```
1 $prodottiDM.onRowComposition = function (row, template)
2 {
3   row.img = "https://source.unsplash.com/200x200/?" +
      this.categoriaImg;
4 };
```

Listato 5.9: Caricamento immagini prodotti

5.5.8 Mappa

Nella sezione dettaglio clienti è possibile visualizzare la mappa dell'indirizzo associato al cliente. Per fare questo è bastato utilizzare l'elemento visuale *Gmap*, presente fra i *Widget* esterni disponibili all'uso all'interno dell'applicazione. Per utilizzare l'oggetto è stato necessario impostare una apiKey sull'oggetto "*app.theme.gmapKey*" per attivare il servizio. Una volta impostato il layout dell'oggetto di classe *Gmap*, è sufficiente impostare le proprietà *address* e *marker* per visualizzare un segnaposto all'interno della mappa, nella località indicata.

```
1 var indirizzo = view.istanzaCliente.indirizzo + "␣" +
     view.istanzaCliente.citta + "␣" + "Italy";
2 $addressContainer.innerText = indirizzo;
3 $gmap.address = indirizzo;
4 $gmap.addMarker("M1", indirizzo);
```

Listato 5.10: Istruzioni per la visualizzazione di un Marker sulla mappa in un certo indirizzo

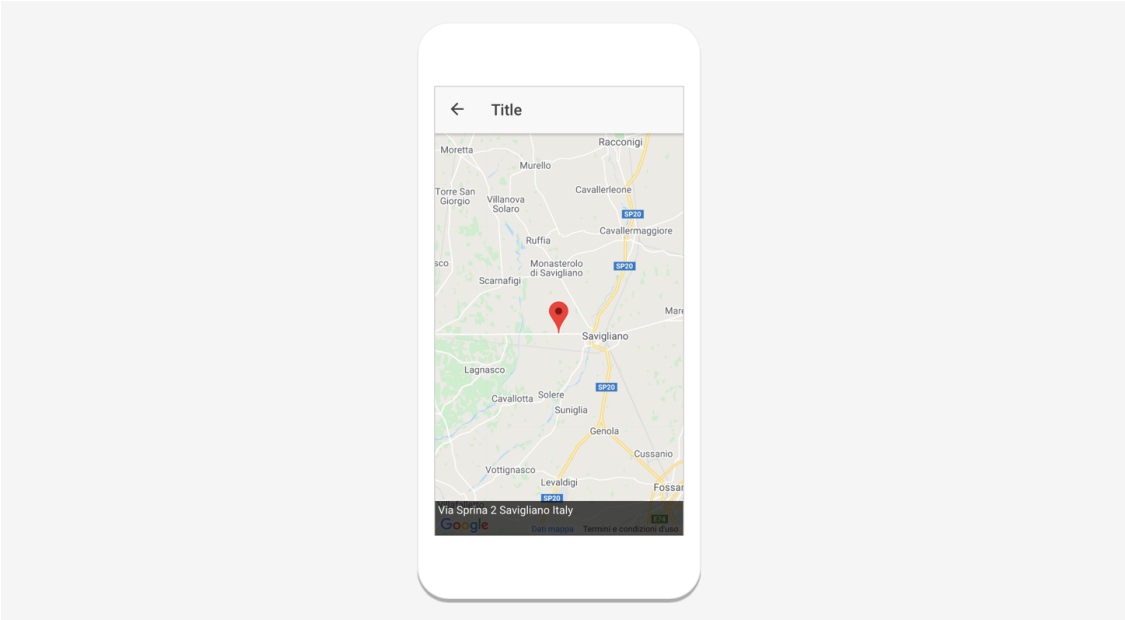

Figura 5.14: Visualizzazione della Gmap per l'indirizzo di un cliente

Capitolo 6

Risultati e conclusioni

La panoramica del mondo delle applicazioni e dei sistemi utilizzabili per il loro sviluppo, mi ha permesso di scegliere con criterio il framework per sviluppare l'applicazione.
Ho trovato estrememante utile la prima parte della tesi, maggiormente analitica, per capire quanto sia importante effettuare una scelta giusta a priori, delineando chiaramente quali sono gli obiettivi e il target richiesto dal software applicativo. Il framework utilizzato è stato scelto appunto in funzione di essi.
Instant Developer Cloud mi ha aiutato in maniera cosiderevole a raggiungere gli obiettivi prefissati, permettendomi di non scendere nello specifico per la gestione dell'interfaccia con gli ambienti nativi di basso livello, ma lasciandomi focalizzare totalmente sullo sviluppo del codice applicativo.
Come discusso ampiamente nei capitoli 4 e 5, non ho dovuto preoccuparmi della portabilità del codice su iOS e Android, ma il framework, con il suo ambiente integrato e i plug-in *Cordova*, mi ha permesso di raggiungere entrambe le piattaforme.
Lo sviluppo di questa tesi ha richiesto circa cinque mesi di lavoro, ma questo tempo non è stato esclusivamente dedicato allo sviluppo. Infatti sono da tenere in considerazione alcuni fattori :

- Approccio allo sviluppo applicativo : questa è stata la prima esperienza personale in cui ho sviluppato una applicazione. Nel primo periodo ho preso conoscenza dell'ambiente, analizzando i principali framework e approcci, in funzione dell'app da sviluppare e dei consigli del mio relatore e tutor aziendale.

- Apprendimento delle tecniche di Instant Developer Cloud : una volta idividuata la piattaforma di sviluppo, è seguito un periodo per l'apprendimento del framework in uso, attravrso corsi ed esempi concreti, resi disponibili dalla piattaforma.

- Sviluppo autonomo : il lavoro è stato svolto in maniera individuale.

- Lavoro Part-time : per il primo periodo non ho lavorato a tempo pieno alla stesura del codice, in quanto ho dovuto alternare lo sviluppo della tesi allo studio dell'ultimo esame.

Mediamente, per lo sviluppo di un'app nativa servono 4,5 mesi di lavoro, che possono variare in funzione della complessità del progetto [16]. Instant Developer Cloud, fra i suoi obiettivi, propone la possibilità di abbassare fino al 70% i tempi di sviluppo, dal design all'immissione dell'app sul mercato.
Tenendo conto dei fattori riportati in precedenza, posso affermare che l'obiettivo riguradante il tempo di sviluppo è stato raggiunto. Dei cinque mesi utilizzati per svolgere la tesi, tre sono stati dedicati all'applicazione, ma di questi tre, soltanto la metà del tempo è stato dedicato allo sviluppo concreto di architettura dati e codice applicativo, utilizzando l'altra metà per l'apprendimento delle tecniche.
I tempi per un progetto futuro potrebbero ridursi ulteriormente, vista l'esperienza che ho acquisito, permettendomi di sviluppare una applicazione mediamente complessa in 1 mese di lavoro indipendente.
Ho potuto quindi dimostrare l'efficacia dell'approccio multipiattaforma orientato alla programmazione in Cloud, che mi ha permesso di raggiungere sia Android che iOS, sia smartphone che tablet, in maniera *full-stack*, e comprensiva quindi di *back-end* e *front-end*, risultato impossibile da ottenere con due sviluppi nativi in parallelo.
Il vantaggio è stato quindi quello di scegliere il framework corretto in funzione del problema che avrei dovuto risolvere. Utilizzando un altro framework non avrei goduto dei vantaggi di tempo, design e portabilità del codice, per lo sviluppo dell'applicazione K-4Sales.
La flessibilità delle funzionalità, per rispondere all'eterogeneità dei clienti, che caratterizzava l'applicazione orginale, è stata tradotta sul prototipo da me sviluppato, ed è stata mantenuta la logica applicativa principale.
Da notare però è il fatto che la versione originale di K-4Sales presenta

alcune funzionalità ulteriori sviluppate negli anni da *Kelyan*, che permettono agli utenti di godere di altri servizi, come una bacheca merketing, l'andamento degli incassi o una sezione dedicata alle statistiche, che non sono state aggiunte nella mia soluzione.
Il primo motivo è per la mancanza di una lincenza professionale del framework *Instant Developer Cloud*, che mi ha limitato nel numero di *View* aggiungibili al progetto.
Il secondo è legato all'obiettivo del prototipo. Il mio scopo non era quello di creare un prodotto finito per rimpiazzare quello già esistente, ma di dimostrare che questo fosse possibile, con un approccio multipiattaforma. Le funzionalità infatti potranno essere estese qualora l'azienda sceglierà di seguire l'approccio da me presentato.
L'approccio Cloud-multipiattaforma allo sviluppo di una applicazione quindi deve essere tenuto in forte considerazione dalle aziende, le quali possono godere di agevolazioni in termini di costi del personale, gestione delle infrastrutture tempi di sviluppo e manutenzione del codice, unici di questa tecnologia.
Dal momento in cui questo apporccio è risultato utile per la tipologia di problema che ho dovuto risolvere, esso potrebbe risultare controproducente qualora i requsiti applicativi fossero estremamente legati alle prestazioni.
Come in tutte le discipline ingegneristiche, anche in questo caso non esiste una soluzione ottima che soddisfa ogni problema, ma ci troviamo nuovamete davanti alla solita risposta del bravo ingegnere: "dipende"!

Bibliografia

[1] ictBusiness.it. *L'app mania continua a crescere*. www.ictbusiness.it/cont/news/l-app-mania-continua-a-crescere-60-di-download-in-due-anni/40808/1.html. 2018.

[2] Ian Blair. *Mobile App Download and Usage Statistics*. https://buildfire.com/app-statistics/. 2019.

[3] Alexander Viken. «The History of Personal Digital Assistants 1980-2000». In: *Agile Mobility* (2009). URL: https://web.archive.org/web/20131030153659/http://agilemobility.net/2009/04/the-history-of-personal-digital-assistants1/.

[4] Statista.com. *Cumulative app download on app store from 2008 to 2017*. www.statista.com/statistics/263794/number-of-downloads-from-the-apple-app-store/. 2019.

[5] Wikipedia. *Google Play*. it.wikipedia.org/wiki/Google_Play.

[6] Enciclopedia Treccani. «Il calcolatore». In: (). URL: www.treccani.it/enciclopedia/calcolatore/.

[7] Danielle Levitas. *The app economy forecast: A 6 trillion dollar market in the making*. www.appannie.com/en/insights/market-data/app-economy-forecast-6-trillion-market-making/. 2017.

[8] Mindsea team. *25 Mobile App Usage Statistics To Know In 2019*. www.mindsea.com/app-stats/. 2019.

[9] www.statista.com.

[10] PoliMi Osservatorio Mobile B2c Strategy. *A tutto Mobile*. www.osservatori.net/it_it/osservatori/comunicati-stampa/a-tutto-mobile-sostituito-pc. 2019.

[11] Mark Dolliver. *US Time Spent with Media in 2019*. www.emarketer.com/content/average-us-time-spent-with-mobile-in-2019-has-increased. 2019.

[12] TheAppSolution.com Valerii Filipets. *Android vs iOS Development*. www.theappsolutions.com/blog/development/ios-vs-android/. 2019.

[13] comScore.com archive. *The 2017 U.S. Mobile App Report*. www.comscore.com/Insights/Presentations-and-Whitepapers/2017/The-2017-US-Mobile-App-Report. 2017.

[14] Wikipedia. *Cloud Computing*. https://it.wikipedia.org/wiki/Cloud_computing.

[15] Wikipedia. *Object-relational mapping*. it.wikipedia.org/wiki/Object-relational_mapping.

[16] Melanie Pinola. *How much time and money it takes to develop a mobile app*. lifehacker.com/this-graphic-explains-how-much-time-and-money-it-takes-1735164869. 2015.