Università degli Studi di Milano Bicocca

Scuola di Scienze

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di laurea in Informatica

Applicazioni Mobili Multipiattaforma - Comparazione Framework di Sviluppo e Realizzazione di un'applicazione con React Native

Relatore: Daniela Micucci

Co-relatore: Davide Ginelli

Relazione della prova finale di:

Luigi Santise

Matricola 844947

Anno Accademico 2020 – 2021

Indice

Introduzione

Il mondo dell'informatica è costantemente in evoluzione: strumenti e tecnologie anche relativamente giovani rischiano ogni giorno di essere superate e migliorate. L'aggiornamento e il rinnovamento sono caratteristiche intrinseche del mondo digitale, e questa tendenza si è ulteriormente accentuata negli ultimi anni, da quando l'informatica è entrata prepotentemente a far parte della vita delle persone.

In particolare, il settore dei dispositivi mobili è uno di quelli che negli ultimi anni è cresciuto in maniera esponenziale, vista la diffusione sempre più imponente delle tecnologie mobili, ed a testimonianza di ciò basta pensare a come erano fatti i primi telefoni cellulari e a quelli che sono invece gli smartphone di oggi. I dispositivi mobili che tutti i giorni vengono utilizzati da utenti di qualsiasi età riescono infatti ad offrire funzionalità che un paio di decenni fa erano praticamente impensabili. Quello della tecnologia mobile è un settore che si sta sempre più andando ad imporre sul mercato, ed è quindi molto importante che chiunque desideri lavorare nel settore informatico abbia almeno una conoscenza di base di quello che è il vastissimo mondo dei dispositivi mobili.

Quando si parla di programmazione di dispositivi mobili si intende la realizzazione di applicazioni mobili. Con il termine "mobile app", solitamente abbreviato molto più semplicemente in "app", si intende un'applicazione software il cui ambiente di esecuzione è quello dei dispostivi mobili, e dunque smartphone e tablet, e che consenta all'utente di fruire di particolari servizi che naturalmente dipendono dal dominio dell'applicazione che si deve creare.

Nell'ambito dei dispositivi mobili, è ovviamente scontato considerare quelli che sono i due principali produttori di sistemi operativi: Android e iOS. Altre tecnologie come ad esempio Windows Phone, KaiOS, Tizen, sono praticamente un granello di sabbia in un deserto rispetto ai due marchi principali, e la quasi totalità del mercato è oggi nelle mani di Google ed Apple. Secondo studi piuttosto recenti [1], aggiornati a giugno 2021, circa il 73% degli utenti utilizza dispositivi Android, circa il 26% utilizza dispositivi iOS, mentre solo una piccolissima parte degli utenti, inferiore all'1% utilizza dispositivi con un sistema operativo differente.

Una situazione di questo genere potrebbe far pensare che gli sviluppatori non abbiano molta scelta sulle tecnologie da utilizzare per realizzare applicazioni per dispositivi mobili. Niente di più sbagliato. Il mondo dei dispositivi mobili è infatti estremamente vasto, ed esistono diverse tipologie di applicazioni mobili, classificate sulla base di come l'applicazione viene concepita ed implementata.

Quando ci si approccia alla programmazione di dispositivi mobili, nella maggior parte dei casi, il punto di ingresso in questo vastissimo mondo è la programmazione di applicazioni native. Si tratta di applicazioni che vengono sviluppate appositamente per poter essere eseguite su una specifica piattaforma, e sono quindi realizzate utilizzando direttamente gli strumenti di sviluppo messi a disposizione dai produttori dei sistemi operativi (Android e iOS).

Sviluppare in nativo comporta indiscutibili vantaggi quali sicuramente la possibilità di sfruttare le API del sistema operativo ed accedere in modo diretto alle funzionalità hardware e software del dispositivo, il che consente di ottenere un livello eccellente per quanto riguarda le prestazioni dell'applicazione, non raggiungibile da qualunque altra tipologia di applicazioni. Dall'altro lato, tuttavia, lo sviluppo di applicazioni native porta con sé anche un numero non indifferente di aspetti negativi.

In particolare, quello che veramente penalizza lo sviluppo di applicazioni native è il fatto che esse sono progettate per essere eseguite su un'unica piattaforma, e dunque, se si sviluppa un'applicazione Android, essa non potrà essere eseguita su un dispositivo iOS e viceversa. Ciò rende di fatto necessario, se si vuole raggiungere sia il mercato Android sia quello iOS, lo sviluppo di due codebase

distinte, il che significa, da un punto di vista pratico, realizzare due diverse applicazioni, una per ciascuna piattaforma.

La necessità di sviluppare due versioni diverse per una stessa applicazione comporta anzitutto la necessità di competenze maggiori per poter sviluppare l'app, e in genere la formazione di un team di sviluppo più ampio ed eterogeneo, composto sia da sviluppatori Android che da sviluppatori iOS, ma ciò che va a impattare in modo negativo sul processo di sviluppo è il fatto che, avendo a che fare con due diverse applicazioni, di fatto quelli che sono i costi, i tempi e le difficoltà di sviluppo, testing, manutenzione e aggiornamento dell'applicazione sono sostanzialmente raddoppiati.

Viene allora spontaneo chiedersi se veramente valga la pena sviluppare applicazioni native, oppure se è possibile in qualche modo aggirare tutte queste difficoltà andando a preferire un approccio diverso per lo sviluppo dell'applicazione, e in particolare andare a scegliere una tecnologia diversa, che consenta di realizzare applicazioni direttamente eseguibili su entrambe le piattaforme. Non esiste una risposta generale a questa domanda: occorre infatti saper scegliere la giusta soluzione in base al contesto in cui si deve sviluppare l'applicazione.

Se aspetti quali le prestazioni e l'ampio utilizzo di funzionalità del dispositivo sono cruciali per quanto riguarda l'applicazione che si deve sviluppare, allora senza dubbio la scelta deve ricadere sulle applicazioni native, perché le applicazioni multipiattaforma non consentono di raggiungere i livelli di efficienza offerti dalle prime. Inoltre, se si desidera trarre profitto dall'applicazione, la scelta deve senza dubbio ricadere sulle applicazioni native, che a parità di funzionalità offrono sicuramente agli utenti l'esperienza più gratificante e un livello di affidabilità e di prestazioni decisamente maggiore rispetto a quello delle applicazioni multipiattaforma.

Se le necessità precedenti vengono meno, e soprattutto se non si hanno risorse e budget sufficienti per poter sviluppare in nativo, dal momento che non si hanno le possibilità per pensare di gestire due codebase distinte, ecco allora che diventa un obbligo prendere in considerazione altre tecnologie di sviluppo.

Nel presente elaborato si andrà ad esplorare il mondo delle applicazioni multipiattaforma e dei framework di sviluppo cross-platform. Nel capitolo 1 si descriveranno le diverse tipologie di applicazioni multipiattaforma, e per ciascuna di esse si farà riferimento ai framework di sviluppo attualmente disponibili. Nel capitolo 2 si andrà a descrivere in maniera più approfondita tre di questi framework, Xamarin, React Native e Flutter, che verranno messi a confronto in modo da far emergere le caratteristiche principali, i pregi e i difetti di ciascuno. Al fine di poter confrontare in modo più approfondito i tre framework, è stata realizzata con ciascuno di essi l'applicazione MyMusic, le cui caratteristiche verranno descritte nel capitolo 3. Ciascun componente del gruppo di lavoro ha realizzato l'applicazione con uno dei tre framework, e il capitolo 4 sarà dedicato a descrivere come MyMusic è stata effettivamente implementata con il framework utilizzato. Nel capitolo 5, infine, verranno confrontate le tre implementazioni, in modo da far emergere aspetti comuni e differenze, nonché problematiche emerse durante lo sviluppo dell'applicazione derivanti dall'utilizzo di un framework piuttosto che di un altro.

1. Applicazioni multipiattaforma (ibride, web native, cross-compiled)

1.1 Lo sviluppo multipiattaforma e quando programmare multipiattaforma

Per far fronte allo svantaggio principale delle applicazioni native, ovvero la necessità di gestire due codebase distinte, con conseguente aumento di tempi e costi di sviluppo, manutenzione e aggiornamento, è possibile scegliere di sviluppare un'applicazione mobile utilizzando l'approccio multipiattaforma. In questo modo, con un'unica base di codice sarà possibile realizzare un'applicazione che potrà essere eseguita direttamente su diversi dispositivi senza tener conto del sistema operativo installato.

Agli occhi di utenti non esperti di tecnologie mobile sarà praticamente impossibile capire se l'app è stata scritta in nativo o se invece è stata realizzata tramite strumenti multipiattaforma. Un'applicazione multipiattaforma progettata e sviluppata in modo opportuno, infatti, può essere eseguita su qualsiasi sistema operativo o dispositivo senza mostrare differenze evidenti rispetto alle classiche applicazioni native. Solamente chi ha sviluppato l'applicazione è davvero a conoscenza delle differenze rispetto al nativo, e delle difficoltà incontrate nello sviluppo ed eventualmente delle limitazioni di alcune funzionalità che non si sono riuscite a superare per garantire il funzionamento su più piattaforme.

La scelta di orientarsi verso lo sviluppo multipiattaforma deve essere presa in forte considerazione ogni qualvolta non si ha a disposizione un budget sufficiente per sostenere i costi e i tempi dello sviluppo nativo. Altro fattore da considerare quando si sceglie di sviluppare un'applicazione è l'importanza di aspetti quali le performance e la necessità di accesso completo alle funzionalità del sistema operativo e del dispositivo: laddove questi aspetti sono secondari, le tecnologie multipiattaforma rappresentano una valida soluzione.

Esistono diverse tecnologie per sviluppare un'applicazione multipiattaforma, tanto che è possibile classificarle in varie tipologie sulla base di come l'applicazione viene realizzata. Si parla quindi a tal proposito di applicazioni ibride, web native e cross-compiled.

La classificazione delle varie tipologie di applicazioni multipiattaforma è ancora un'area piuttosto grigia. Infatti, le tecnologie che oggi si hanno a disposizione sono talmente numerose che posizionarle tutte su una scala gerarchica ben definita è un compito tutt'altro che semplice, anche perché si tratta di un mondo in continua evoluzione ed espansione.

1.2 Applicazioni Ibride

La prima tipologia di applicazioni mobili multipiattaforma è quella delle applicazioni ibride (hybrid app), che a livello logico possono essere poste nel livello più alto della classificazione.

Quando si parla di tecnologie ibride si intende la realizzazione di un'applicazione che di fatto corrisponde ad una via di mezzo tra lo sviluppo mobile vero e proprio e quello web. Gli strumenti di sviluppo ibrido consistono infatti sostanzialmente nell'implementazione di codice basato su tecnologie web, quali HTML, CSS, JavaScript, che viene poi incorporato ed eseguito all'interno di un contenitore nativo, che di fatto realizza il passo decisivo per passare dallo sviluppo web a quello mobile, consentendo agli utenti un'esperienza pressoché identica a quella dell'utilizzo di una classica applicazione.

In particolare, le tecnologie ibride si basano su un particolare componente nativo in grado di adattare alla piattaforma mobile il contenuto dell'applicazione. Ogni piattaforma necessita dello specifico contenitore nativo. Per esempio, per Android viene utilizzato WebView, mentre in iOS si usa WKWebView.

1.2.1 Vantaggi delle applicazioni ibride

Il principale vantaggio delle applicazioni ibride è sicuramente l'aspetto multipiattaforma: al contrario delle app native, che devono essere programmate per ciascun sistema operativo, è sufficiente realizzare un'unica applicazione, che a quel punto risulta pronta per essere eseguita su qualsiasi piattaforma. L'utilizzo delle tecnologie ibride per realizzare un'applicazione consente in questo modo di abbattere i tempi e i costi di realizzazione dell'applicazione rispetto a quelli necessari per lo sviluppo nativo.

Un vantaggio notevole dello sviluppo ibrido consiste poi nel fatto che, una volta sviluppata un'applicazione ibrida, si ha già a disposizione codice che potrà essere semplicemente adattato per ottenere anche la versione web dell'applicazione, e ovviamente vale anche il viceversa: se si ha già sviluppato un'applicazione web, è decisamente semplice trasformarla in un'applicazione ibrida.

Le applicazioni ibride rappresentano a livello logico il primo passo per passare dallo sviluppo web a quello di applicazioni mobili vere e proprie. Come tali, introducono una serie di possibilità non raggiungibili con l'approccio web puro.

Esse possono essere eseguite, nei limiti delle funzionalità implementate, anche senza avere a disposizione una connessione Internet, e quindi in modalità offline. Le applicazioni ibride consentono poi un ampio accesso alle funzionalità hardware e software del dispositivo mobile (fotocamera, microfono, dati, notifiche push, GPS, sensori per la geolocalizzazione). Esse, inoltre, sono pubblicate negli app store ufficiali delle varie piattaforme, e consentono quindi di raggiungere una grande quantità di utenti mobili.

Dal punto di vista implementativo, le tecnologie ibride sono infine piuttosto semplici da imparare e da utilizzare, ed è inoltre possibile integrare una qualsiasi libreria o framework JavaScript agli specifici strumenti utilizzati per sviluppare l'applicazione.

1.2.2 Svantaggi delle applicazioni ibride

Naturalmente, le tecnologie ibride non sono perfette e comportano, accanto a una serie di indiscutibili vantaggi, anche alcuni aspetti negativi.

Per prima cosa, anche se le applicazioni ibride offrono prestazioni migliori rispetto a quelle web, si tratta comunque di performance non paragonabili a quelle delle applicazioni native. Per interfacciarsi con i componenti nativi, infatti, le applicazioni ibride utilizzano dei livelli di software intermedio (bridge), il che rende decisamente più pesante il processo di computazione.

Sulla base di ciò, come accade per tutte le applicazioni multipiattaforma, se si sviluppa un'applicazione ibrida, a causa della separazione tra l'implementazione software e l'hardware nativo, sarà molto difficile ottimizzarla per una specifica piattaforma, che sia iOS o Android.

Infine, l'accesso alle funzionalità hardware e software del dispositivo è ancora incompleto, come conseguenza intrinseca della lontananza delle tecnologie ibride da quelle native.

1.2.3 Quando scegliere l'approccio ibrido

Visti gli aspetti negativi che le tecnologie ibride comportano, occorre allora cercare di capire quando esse rappresentano la soluzione migliore per lo sviluppo di un'applicazione.

Le tecnologie ibride non sono adatte se occorre sviluppare applicazioni particolarmente complesse. Se infatti le funzionalità da implementare iniziano ad essere tante e soprattutto se esse richiedono l'accesso a varie funzionalità del dispositivo, l'applicazione non solo risulterà decisamente lenta, ma comporterà anche un grande utilizzo di memoria, a causa della presenza della WebView, che effettua il rendering di tutti i componenti.

Al contrario, è opportuno decidere di usare le tecnologie ibride se l'applicazione da sviluppare è relativamente semplice e se si ritiene che una web app non sia sufficiente per ciò che si desidera realizzare. In particolare, le tecnologie ibride sono perfette se si è nella situazione in cui, con un budget limitato, si vuole creare, a partire da un sito web, una versione mobile da distribuire velocemente agli utenti iOS e Android.

In ogni caso, se le prestazioni dell'applicazione sono un aspetto cruciale a cui non si può rinunciare, è al contrario consigliato lasciare perdere le soluzioni ibride e spostarsi sulle applicazioni native, o almeno, se non si hanno le risorse necessarie, sull'approccio web nativo o cross-compiled.

L'approccio ibrido rappresenta una valida alternativa per lo sviluppo di applicazioni mobili. A testimonianza di ciò, basti pensare che applicazioni molto note, come Gmail, Twitter e Amazon, sono realizzate con tecnologie ibride.

1.2.4 Tecnologie e framework per lo sviluppo di applicazioni ibride

Per quanto riguarda le tecnologie ibride, sono disponibili una serie di framework e strumenti di sviluppo. Se si decide di implementare un'applicazione utilizzando l'approccio ibrido, la scelta del framework da utilizzare probabilmente ricadrà su uno degli strumenti presenti nella seguente lista, che comprende quelli attualmente più diffusi ed utilizzati.

- Ionic [47] – Si tratta di un framework open-source ed è uno dei più utilizzati dagli sviluppatori. Basato interamente su tecnologie Web quali HTML5, CSS e SASS, Ionic nasce dalla combinazione di AngularJS per la realizzazione delle interfacce grafiche e Cordova, che consente alle app realizzate tramite Ionic di essere installate sui dispositivi mobili. I suoi principali punti di forza sono proprio la realizzazione degli aspetti grafici dell'applicazione, oltre all'ampio accesso alle funzionalità native dei dispositivi e alla curva di apprendimento decisamente rapida.
- Apache Cordova (PhoneGap) [48] – Nato dal suo antenato PhoneGap, è ancora uno dei più diffusi nonostante non sia così recente. Basato sulle classiche tecnologie web (HTML5, CSS, JavaScript), Apache Cordova è il framework per lo sviluppo ibrido che in assoluto consente la maggior affidabilità di compatibilità tra le varie piattaforme, è semplice da imparare e consente ampio accesso a funzionalità native.
- Framework7 [49] – Si tratta di un framework HTML mobile open source per lo sviluppo di applicazioni ibride, web app e PWA con aspetto e funzionalità nativi. Si concentra esclusivamente sullo sviluppo per Android e iOS. È facile da imparare, contiene molte librerie di supporto integrate, e una serie molto ampia di widget e componenti, il che lo rende una buona scelta per lo sviluppo di applicazioni.
- Kendo UI [50] – Si tratta di un framework open-source, ma rivolto esclusivamente a clienti aziendali, per cui attualmente non ne esiste una versione gratuita. L'interfaccia utente di Kendo si propone essenzialmente di fornire una raccolta ampia di componenti dell'interfaccia utente JavaScript, integrandoli con librerie per jQuery, Angular, Vue, React, per consentire ai team di sviluppo aziendali di realizzare applicazioni mobili ibride con prestazioni elevate.
- Onsen UI [51] – Si tratta di un framework gratuito, molto flessibile e semplice da utilizzare. L'interfaccia utente Onsen consente di scegliere tra diversi framework (AngularJS, Angular, React, Vue) oppure di passare al linguaggio JavaScript puro per lo sviluppo di un'applicazione ibrida. Onsen mette a disposizione una serie davvero ampia di componenti pronti da utilizzare.

Come si può intuire, la lista è veramente molto ampia e in continuo aggiornamento. Fra i framework disponibili per lo sviluppo di applicazioni ibride è possibile citare anche Quasar [52], Aurelia [53], ExtJS [54], Capacitor [55].

1.3 Applicazioni Web Native

La seconda tipologia di applicazioni mobili multipiattaforma è quella delle applicazioni web native (web native app), che da un punto di vista logico possono essere poste nel livello intermedio della classificazione, fra le applicazioni ibride e quelle cross-compiled.

Nella classificazione delle applicazioni multipiattaforma, quello delle applicazioni web native è un livello nuovo e soprattutto molto sottile che spesso viene ignorato. Non è infatti raro trovarlo incorporato nel livello delle applicazioni ibride o in quello delle applicazioni cross-compiled.

Le tecnologie che talvolta vengono indicate come "web native" sono quelle in cui si realizza il punto di incontro vero e proprio tra le tecnologie web e quelle native.

La particolarità che differenzia questa tipologia di applicazioni da quelle ibride è la mancanza della WebView e quindi di un componente che interpreti direttamente il codice sviluppato in JavaScript sul contenitore nativo.

Per poter essere eseguito, il codice subisce una serie di trasformazioni che però non sono ancora sufficienti per far sì che esso possa essere compreso dai sistemi operativi Android o iOS. Infatti, a livello nativo, non sono disponibili componenti in grado di interpretare il codice JavaScript, ad eccezione della WebView che però appesantisce la computazione. L'idea alla base delle tecnologie web native è allora quella di fornire, insieme al codice, anche il motore JavaScript. Il passo che manca è l'intervento del bridge, uno strato software intermedio che realizza il ponte tra le tecnologie JavaScript e i componenti nativi.

1.3.1 Vantaggi delle applicazioni Web Native

L'utilizzo delle tecnologie web native introduce una serie decisamente ampia di aspetti positivi.

Naturalmente le applicazioni web native sono multipiattaforma, ed è quindi possibile, con un'unica codebase, raggiungere gli utenti di tutte le piattaforme. Ciò consente, come accadeva per le applicazioni ibride, di ridurre i costi e i tempi di realizzazione, manutenzione e aggiornamento dell'applicazione.

Nonostante la presenza del bridge, l'approccio web nativo consente di raggiungere prestazioni nettamente migliori rispetto all'ibrido, che iniziano ad avvicinarsi a quelle delle applicazioni native, dal momento che i componenti utilizzati sono gli stessi o comunque sono basati sulle stesse tecnologie di quelle dei componenti delle applicazioni native.

Infine, nonostante siano necessari alcuni aggiustamenti, il codice di un'applicazione web nativa è ancora quasi del tutto compatibile con quello di un'applicazione web, per cui è molto semplice ottenere la versione web a partire da un'applicazione web nativa o viceversa trasformare un'applicazione web nativa in una web app.

1.3.2 Svantaggi delle applicazioni Web Native

Aldilà degli indiscutibili vantaggi di questa particolare tipologia di applicazioni, vanno sottolineati anche alcuni aspetti negativi.

Sebbene la maggioranza del codice sia condiviso tra le varie piattaforme, la realizzazione di alcune funzionalità più complesse potrebbe comportare la necessità di scrivere del codice nativo specifico

per ogni singola piattaforma. In questi casi, gli sviluppatori dovranno essere a conoscenza di alcune caratteristiche dei componenti nativi, in modo da poterli utilizzare in maniera opportuna. Ciò potrebbe causare difficoltà non indifferenti per gli sviluppatori web che desiderano realizzare un'applicazione mobile sfruttando le tecnologie web native.

Inoltre, la curva di apprendimento delle tecnologie web native risulta in genere essere più ripida rispetto a quella delle tecnologie ibride o web, specialmente per quanto riguarda gli aspetti dell'interfaccia utente e del layout, per cui anche programmatori web esperti potrebbero trovare non poche difficoltà.

Da ultimo ma non meno importante, la presenza del bridge non consente di arrivare a prestazioni paragonabili a quelle delle applicazioni cross-compiled e native.

1.2.3 Quando scegliere l'approccio web nativo

In generale, è bene realizzare un'applicazione tramite strumenti di sviluppo che sfruttino l'approccio web nativo quando si desidera realizzare un'applicazione multipiattaforma che abbia buone prestazioni, per cui l'approccio ibrido non sarebbe sufficiente.

Naturalmente, se le prestazioni richieste devono avvicinare quelle native, probabilmente sarebbe meglio orientarsi verso la soluzione cross-compiled.

Se tuttavia si ha a disposizione un team di sviluppo abituato a lavorare con tecnologie web e non con linguaggi di programmazione specifici, e quindi se l'approccio web nativo consente di abbattere quelli che sono i tempi e i costi di realizzazione dell'applicazione, l'utilizzo di tecnologie simili è decisamente un buon compromesso che può e anzi deve essere tenuto in considerazione.

L'approccio web nativo rappresenta una buona scelta per lo sviluppo di applicazioni mobili. A testimonianza di ciò, basti pensare che applicazioni molto note, tra cui Facebook, Instagram e Skype, sono realizzate con tecnologie web native.

1.2.4 Tecnologie e framework per lo sviluppo di applicazioni web native

Come per le applicazioni ibride, esistono una serie di strumenti di sviluppo per realizzare applicazioni web native. I framework oggi più diffusi sono contenuti nella seguente lista.

- React Native [22] – Si tratta di uno dei migliori framework di sviluppo di applicazioni mobili multipiattaforma ed è uno dei più diffusi in assoluto. Supportato da Facebook, il framework è open source e consente di creare applicazioni mobili multipiattaforma con il design tipico del framework web React. Consente di creare, tramite JavaScript, applicazioni mobili assimilabili a quelle create tramite approccio nativo.
- Appcelerator Titanium [56] – Si tratta di un framework per applicazioni mobili basate su JavaScript. Rispetto alle applicazioni ibride, il codice implementato viene compilato e costituisce di fatto una combinazione fra JavaScript e il linguaggio nativo. Tutto ciò migliora notevolmente le prestazioni rispetto al classico sviluppo ibrido. Appcelerator Titanium costituisce una buona alternativa a React Native per lo sviluppo di applicazioni web native.
- NativeScript [57] – Si tratta anche in questo caso di una delle principali soluzioni per lo sviluppo web nativo, e quindi adatta a tutti coloro che desiderano sviluppare applicazioni multipiattaforma utilizzando tecnologie web. Le app realizzate attraverso NativeScript sono di fatto indistinguibili da quelle native, e utilizzano le stesse API usate nei classici ambienti di sviluppo per applicazioni native, Android Studio e Xcode.

- Weex [58] – Un altro framework da citare in questa categoria è senza dubbio Weex. Si tratta anche in questo caso di uno strumento che consente di creare applicazioni simili a quelle native utilizzando tecnologie web, e in particolare con Weex si utilizzano i comuni tag HTML e i comandi CSS per richiamare componenti nativi.
- Svelte Native [59] – Si tratta di uno dei più giovani framework per lo sviluppo di applicazioni web native. Si basa sul framework web Svelte, che sta crescendo in popolarità tra gli sviluppatori web grazie alla sua semplicità, e su NativeScript. Mentre i framework di applicazioni web native come React Native eseguono la maggior parte del lavoro durante l'esecuzione vera e propria dell'applicazione e quindi sul dispositivo mobile, l'approccio seguito da Svelte Native è decisamente nuovo, e consiste nello spostare la maggior parte del lavoro nella fase di compilazione.

Nel classificare le applicazioni multipiattaforma, molto spesso si tende a unire le applicazioni ibride e quelle web native in unico gruppo, così come è frequente trovare le applicazioni web native insieme a quelle cross-compiled. Non è quindi strano che, online, si trovino insieme anche i framework utilizzati per lo sviluppo di applicazioni che sfruttano queste tecnologie. I framework qui indicati come web nativi potrebbero quindi essere anche classificati come ibridi o cross-compiled a seconda della classificazione che si sceglie di seguire.

1.4 Applicazioni cross-compiled

L'ultima tipologia di applicazioni multipiattaforma è quella delle applicazioni cross-compiled (cross-compiled app), che a livello logico possono essere poste nel livello più basso della classificazione, e quindi sono quelle più vicine alle applicazioni native.

È al livello delle applicazioni cross-compiled che ci si stacca completamente da quello che è il mondo dello sviluppo tramite tecnologie web. Quando uno sviluppatore realizza un'applicazione con tecnologia cross-compiled, quello che sostanzialmente fa è utilizzare un framework specifico che consenta di scrivere codice in un linguaggio di programmazione dedicato. Il codice verrà poi automaticamente compilato e ottimizzato per le varie piattaforme.

1.4.1 Vantaggi delle applicazioni cross-compiled

Come per tutte le tipologie di applicazioni multipiattaforma, lo sviluppo di un'applicazione cross-compiled avviene tramite un'unica codebase con la quale si raggiungono tutte le piattaforme, e ciò consente di abbattere i tempi e i costi di realizzazione rispetto a quelli delle applicazioni native.

Rispetto alle applicazioni ibride e web native, le applicazioni cross-compiled migliorano in modo importante quelle che sono le prestazioni dell'applicazione. Essendo infatti le tecnologie multipiattaforma più simili a quelle native, si arriva con le applicazioni cross-compiled ad ottenere prestazioni davvero simili a quelle delle applicazioni native. Ciò deriva dal fatto che il codice scritto dagli sviluppatori viene automaticamente compilato in codice nativo, senza bisogno di componenti aggiuntivi (la WebView per le hybrid app) o di uno strato software ulteriore (il bridge per le applicazioni web native).

1.4.2 Svantaggi delle applicazioni cross-compiled

Naturalmente, le applicazioni cross-compiled non sono perfette, e ci sono aspetti negativi che vanno considerati quando si parla di questa tipologia di applicazioni.

Rispetto a quanto accadeva per le app ibride e web native, il codice di un'applicazione cross-compiled non è più in nessun modo compatibile con lo sviluppo web.

Nel caso in cui vengano rilasciati aggiornamenti software dalle varie piattaforme, non sarà immediatamente possibile supportare le nuove funzionalità aggiunte se si programma multipiattaforma, ma occorrerà attendere che il framework di sviluppo cross-compiled utilizzato venga a sua volta aggiornato.

Non va poi dimenticato che, in alcuni casi, al fine di garantire il funzionamento multipiattaforma per l'applicazione, potrebbe essere necessario scrivere delle parti di codice in nativo. Questo fa sì che chi sviluppa multipiattaforma debba comunque avere una minima conoscenza di quelle che sono le tecnologie e i componenti nativi.

Da un punto di vista dell'accesso alle funzionalità native del sistema operativo e del dispositivo, se si sviluppa con strumenti cross-compiled sarà più difficile avere accesso completo a tali funzionalità.

Inoltre, dal punto di vista dell'interfaccia e dell'esperienza utente, il fatto di avere un'unica applicazione che verrà eseguita su entrambe le piattaforme potrà causare, in certi casi, un'interfaccia o un'esperienza utente più o meno diversa e quindi in un certo senso strana per quelle che sono le abitudini degli utenti della specifica piattaforma.

Dal punto di vista delle prestazioni, infine, tramite l'approccio cross-compiled è decisamente difficile ottimizzare un'applicazione sviluppata con una sola codebase per il funzionamento ottimale sulle diverse piattaforme. Sebbene le performance di un'applicazione cross-compiled si avvicinino molto a quelle native rispetto a tutte le altre tipologie di applicazioni multipiattaforma, comunque si tratta di prestazioni ancora inferiori rispetto a quelle delle applicazioni native.

1.4.3 Quando scegliere l'approccio cross-compiled

Sulla base degli aspetti positivi e negativi che caratterizzano lo sviluppo di applicazioni mobili tramite tecnologie cross-compiled, è lecito domandarsi quando è opportuno adottare questo approccio e quando invece orientarsi verso altre soluzioni.

Prima di tutto, va ribadito che, in assenza di limiti di risorse e budget, e nei casi in cui si desidera realizzare applicazioni con le massime prestazioni possibili, la scelta deve ricadere sulle applicazioni native. Se tali necessità vengono meno, è chiaro che la scelta delle applicazioni cross-compiled rappresenta un notevole compromesso tra quello che è l'approccio multipiattaforma e le prestazioni.

L'approccio cross-compiled consente infatti di realizzare un'applicazione caratterizzata da una UI (User Interface) e una UX (User Experience) che segua in modo più o meno preciso le linee guida delle varie piattaforme e quindi risulti non perfetta ma quantomeno soddisfacente per i loro utenti.

Quando si prende in considerazione l'approccio cross-compiled, va sottolineato che si tratta di una soluzione in genere migliore di quella ibrida, ma è anche vero che tale soluzione non consente di utilizzare tecnologie web per lo sviluppo. Quindi, se si ha a disposizione un team esperto in tecnologie web, per ridurre i costi e i tempi di realizzazione ha senso decidere di preferire lo sviluppo ibrido o web nativo rispetto a quello cross-compiled.

Se poi si ha l'intenzione di trarre guadagno dall'applicazione, è bene scegliere di orientarsi sull'approccio nativo. Infatti, se si sviluppa un'applicazione multipiattaforma che offre determinate funzionalità, non è da escludere che sul mercato vi siano già altre applicazioni che offrano tali servizi sviluppate in nativo, e quindi in grado di fornire prestazioni e affidabilità migliori, nonché funzionalità più ampie rispetto alla stessa applicazione sviluppata con approccio multipiattaforma.

L'approccio cross-compiled rappresenta una buona scelta per lo sviluppo di applicazioni mobili. A testimonianza di ciò, basti pensare che applicazioni molto note, tra cui Ebay, UPS, Groupon sono realizzate con queste tecnologie.

1.4.4 Tecnologie e framework per lo sviluppo cross-compiled

Anche per quanto riguarda lo sviluppo cross-compiled, sono disponibili attualmente diversi framework e strumenti di sviluppo che consentono di realizzare applicazioni con tale approccio. Si tratta anche in questo caso di strumenti praticamente appena nati e in continuo sviluppo. La seguente lista comprende quelli attualmente più diffusi e preferiti dagli sviluppatori.

- Flutter [30] – Sviluppato, supportato e gestito da Google, Flutter è probabilmente quello che viene considerato il migliore fra gli strumenti di sviluppo multipiattaforma. Esso consente di creare applicazioni multipiattaforma utilizzando il linguaggio di programmazione Dart. Il punto di forza di Flutter, oltre a una curva di apprendimento decisamente rapida e a una disponibilità enorme di plugin, è la presenza dei widget, elementi di interfaccia utente riutilizzabili e personalizzabili a seconda delle esigenze degli sviluppatori, e basati sulle linee guida delle piattaforme iOS (Cupertino) e Android (Material design).
- Xamarin [15] – Il framework è open-source e di proprietà di Microsoft e consente di realizzare applicazioni mobili multipiattaforma. Basato su C# e .NET, Xamarin è senza dubbio uno dei framework più utilizzati in assoluto dagli sviluppatori. Xamarin consente di realizzare applicazioni mobili programmando in C# e mette a disposizione degli utenti una serie di potenti librerie per accedere ad API native e di terze parti che consentono di sfruttare in modo pressoché completo le funzionalità dei dispositivi mobili.
- RubyMotion [60] – Si tratta di un'altra soluzione interessante per lo sviluppo cross-platform. RubyMotion è un framework open-source che consente di scrivere applicazioni multipiattaforma utilizzando il linguaggio Ruby, molto dinamico e semplice da imparare. Le applicazioni sviluppate con questo framework sono praticamente indistinguibili da quelle native.
- Solar2D (Corona SDK) [61] – È un framework di sviluppo di applicazioni mobili multipiattaforma basato sul linguaggio Lua, molto leggero e multi-paradigma. Solar2D, che sta sostituendo il vecchio Corona SDK, è uno strumento che si concentra sugli elementi principali dello sviluppo di applicazioni mobili: facilità d'uso, velocità, estensibilità, portabilità e scalabilità. Il framework è completamente gratuito e funziona in modo simile a Flutter, e, proprio come il framework di Google, fornisce una grande quantità di plugin agli sviluppatori per qualsiasi tipo di esigenza.

2. Framework scelti per realizzare un'applicazione al fine di confrontarli

Vista la grande varietà di approcci e di framework di sviluppo disponibili attualmente per realizzare un'applicazione multipiattaforma, viene spontaneo chiedersi, quando si inizia a entrare nel mondo dello sviluppo cross-platform, quale sia il framework migliore.

Se da una parte è vero che ogni framework non è perfetto, e che quindi ognuno di essi avrà dei punti di forza e delle debolezze, dall'altra parte è anche vero che, al momento, alcuni sono molto più utilizzati rispetto ad altri in quanto, grazie alle loro potenzialità, sono diventati quelli preferiti dagli sviluppatori. In particolare, effettuando ricerche online [11, 12, 13], quando si parla dei framework multipiattaforma più diffusi, fra i nomi più ricorrenti rientrano senza dubbio Xamarin [15], React Native [22] e Flutter [30].

Questi tre framework sono stati quindi scelti per lo sviluppo di un'applicazione, in modo da confrontarli, facendo emergere le differenze tra essi e mettendo in luce le potenzialità di ciascuno.

2.1 Popolarità e diffusione dei framework

Nel corso degli anni la popolarità e la diffusione dei tre framework si sono modificate, in relazione ai loro aggiornamenti e sviluppi. Naturalmente, essendo molto più giovani, React Native e Flutter sono fenomeni abbastanza recenti e che negli ultimi anni hanno registrato un notevole successo.

Xamarin, nato nel 2011, si è presto affermato come primo framework multipiattaforma. Il dominio di Xamarin si è prolungato per diversi anni, ma in tempi più recenti sono stati rilasciati nuovi framework, che hanno iniziato ad attirare l'interesse degli sviluppatori mobili.

In particolare, il successo maggiore è stato ottenuto da React Native, che nel 2017 ha superato Xamarin imponendosi come il framework multipiattaforma più utilizzato, e soprattutto da Flutter, che dopo aver rapidamente sorpassato Xamarin a pochi mesi dalla sua nascita, ha addirittura superato React Native, diventando nel 2019 lo strumento di sviluppo cross-platform preferito dagli sviluppatori.

La figura 2.1 mostra l'andamento delle ricerche Google in tutto il mondo negli ultimi cinque anni relativamente ai nomi dei tre framework (le ricerche sono quantificate in una scala da 0 a 100).

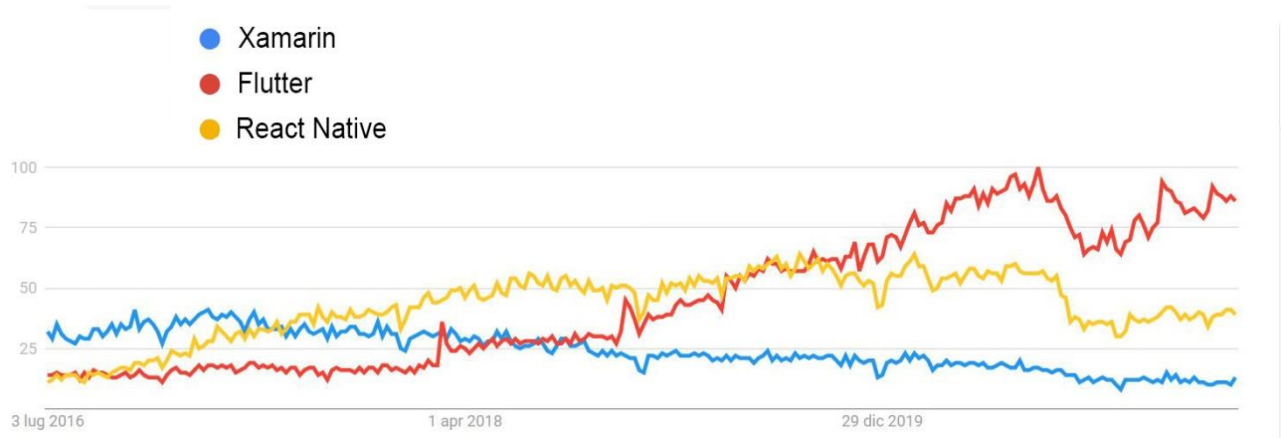

Figura 2.1: Popolarità di Xamarin, Flutter e React Native

Nonostante la diffusione eccezionale registrata da Flutter e React Native negli ultimi cinque anni, Xamarin è riuscito comunque a mantenere un bacino d'utenza piuttosto ampio, e si tratta di un risultato che, a dieci anni ormai dalla nascita del framework, dimostra la sua affidabilità e la sua efficacia.

2.2 Xamarin

Xamarin è il framework più datato fra i tre scelti, nato nel 2011 grazie ai due fondatori Nat Friedman e Miguel De Icaza, che già avevano collaborato alla realizzazione di Mono, un framework open source che era stato realizzato con lo scopo di implementare .NET su sistemi Linux. La figura 2.2 mostra il logo del framework.

Figura 2.2: Logo di Xamarin

All'epoca i framework multipiattaforma presenti sul mercato non erano molti, e così nacque l'idea di creare uno strumento per realizzare applicazioni cross-platform utilizzando un unico linguaggio orientato agli oggetti. A partire da Mono, a tal proposito, fu così realizzato Xamarin, basato sul linguaggio C#.

Nato come azienda indipendente, il progetto Xamarin riscosse un successo tale da suscitare l'interesse di Microsoft, che la acquisì nel 2016. Da quel momento, Xamarin si impose come framework gratuito e open source, ed offre agli sviluppatori una valida alternativa per sviluppare applicazioni multipiattaforma.

2.2.1 I Due Approcci di Xamarin

C# è un linguaggio di programmazione dinamico e orientato agli oggetti. Essendo molto simile a C++ e Java, risulta molto semplice da imparare per programmatori che utilizzano tali linguaggi. Rispetto a C e C++, esso introduce funzionalità importanti, fra cui su tutte la presenza del Garbage Collector, che consente al programmatore di lasciare al compilatore la gestione della memoria allocata.

Xamarin utilizza questo linguaggio per consentire agli sviluppatori di realizzare applicazioni perfettamente funzionanti sia per Android che per iOS. In particolare, è possibile utilizzare il framework di Microsoft scegliendo fra due diversi approcci, quello classico e quello basato sulla più recente tecnologia Xamarin.Forms. Le caratteristiche delle due possibilità di utilizzo di Xamarin sono riassunte nella figura 2.3.

La modalità di utilizzo tradizionale, supportata fin dalle prime versioni del framework, separa la realizzazione della logica dell'applicazione e quella dell'interfaccia utente. Lo sviluppo della parte logica viene realizzato in C# ed è comune per tutte le piattaforme, mentre l'interfaccia utente deve essere specifica per ciascuna piattaforma.

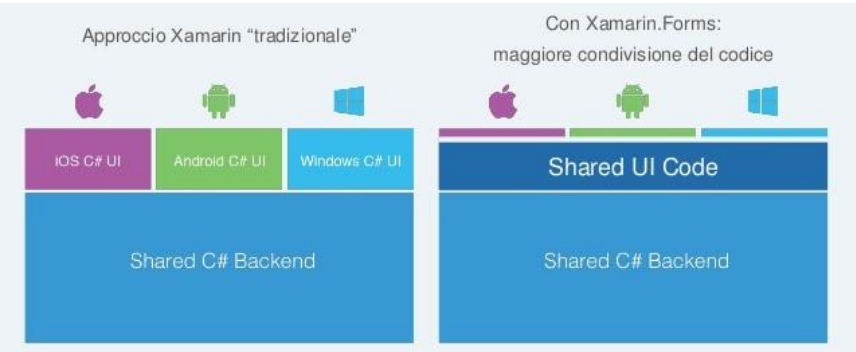

Figura 2.3: I due approcci di Xamarin

Il programmatore che sceglie di utilizzare l'approccio classico deve così conoscere anche le tecnologie native Android e iOS. Se si vogliono raggiungere entrambe le piattaforme, infatti, occorre creare due progetti separati, utilizzando Xamarin.Android e Xamarin.iOS. Nella maggior parte dei casi comunque l'80% del codice dell'applicazione è condiviso tra i due progetti e solo le parti di interfaccia utente verranno realizzate con XML (Android) e XIB (iOS).

Dal 2014 è stata introdotta anche la seconda modalità di utilizzo di Xamarin, chiamata Xamarin.Forms, attraverso cui venne consentita la condivisione totale del codice per tutte le piattaforme, non più solo per la logica applicativa, ma anche a livello di interfaccia utente, che viene realizzata utilizzando XAML oppure C#. In fase di esecuzione i componenti responsabili del rendering si occuperanno di scegliere i corrispettivi elementi nativi in base alla piattaforma su cui viene eseguita l'applicazione.

L'approccio Xamarin.Forms, pur essendo più semplice da utilizzare per il programmatore, non è perfetto. In alcuni casi non è possibile utilizzare componenti specifici, perché con Xamarin.Forms si possono sfruttare solamente quelli che trovano un corrispondente in entrambe le piattaforme.

Inoltre, alcune funzionalità native necessitano di essere gestite in maniera differente a seconda della piattaforma, essendo basate su API completamente diverse. In parziale soccorso è stata recentemente realizzata la libreria Xamarin.Essentials, che consente l'accesso ad alcune di esse.

Xamarin.Forms è un approccio giovane e ancora in evoluzione, per cui non sempre si potranno utilizzare funzionalità, librerie o componenti presenti in altri contesti di sviluppo. Per installare plugin di terze parti è possibile utilizzare NuGet, il package manager gratuito tipico delle piattaforme Microsoft.

Per tutti questi motivi, il nuovo approccio offerto da Xamarin.Forms non ha ancora sostituito completamente quello tradizionale, tanto che molti sviluppatori preferiscono ancora utilizzare Xamarin.Android e Xamarin.iOS.

2.2.2 Compilazione ed Esecuzione

Una volta realizzata l'applicazione tramite C#, per poter ottenere codice eseguibile sarà ovviamente necessario effettuare un processo di compilazione, che Xamarin gestisce in modo diverso a seconda dell'ambiente di esecuzione dell'applicazione, Android o iOS.

La compilazione dei progetti Xamarin.Android inizia con una fase di trasformazione del codice C# in un linguaggio intermedio (Intermediate Language – IL). A questo punto, all'avvio dell'applicazione, il codice IL ottenuto viene compilato in modalità JIT (Just-In-Time) e trasformato in codice assembly nativo direttamente eseguibile.

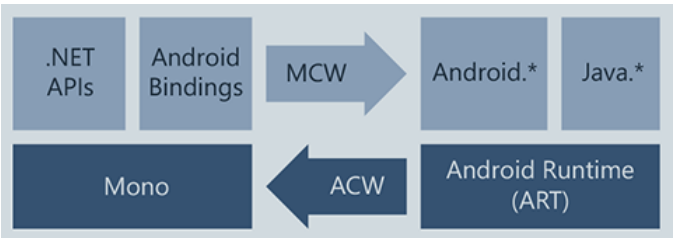

Figura 2.4: Compilazione ed esecuzione Xamarin.Android

Il contesto di esecuzione Android di Xamarin, come mostrato nella figura 2.4, è basato sull'interazione fra l'ambiente Mono, il sistema su cui Xamarin stesso è fondato, e la macchina virtuale Android Runtime (ART).

Trattandosi di ambienti basati su tecnologie differenti, i due contesti di esecuzione devono poter comunicare.

A tale scopo, Xamarin.Android realizza il binding attraverso un meccanismo chiamato Managed Called Wrapper (MCW), di fatto un bridge tra ambiente Microsoft e interfaccia Android. Anche l'ambiente nativo Java, e quindi ART, deve poter interfacciarsi con Mono, e lo fa attraverso il bridge denominato Android Callable Wrapper (ACW).

I progetti Xamarin.iOS vengono invece compilati completamente in modalità AOT (Ahead Of Time), e dunque il codice C# viene direttamente trasformato in codice assembly nativo prima della fase di esecuzione.

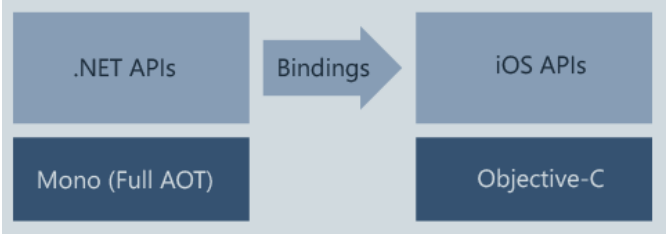

Figura 2.5: Compilazione ed esecuzione Xamarin.iOS

Per passare dall'ambiente Microsoft Mono, codice C#, all'ambiente iOS, codice nativo Objective-C, uno strato software intermedio effettua il binding fra componenti e procedure .NET trasformandole in chiamate alle API native iOS. Si ottiene come risultato un file binario iOS nativo, pronto per essere eseguito. Il processo di compilazione ed esecuzione dei progetti Xamarin.iOS è sintetizzato nella figura 2.5.

Per quel che riguarda l'approccio Xamarin.Forms, quel che accade sostanzialmente è che sulla base della piattaforma target viene selezionato il processo di compilazione opportuno. Se la piattaforma è Android, si segue il procedimento di compilazione per Xamarin.Android, mentre se si esegue l'applicazione su iOS verrà seguita la procedura di compilazione per Xamarin.iOS.

2.3 React Native

React Native è l'unico fra i tre framework scelti che si posiziona in una categoria di applicazioni multipiattaforma diversa da quella cross-compiled. Infatti, quelle realizzate con React Native sono applicazioni classificate come web native. La figura 2.6 mostra il logo del framework.

Figura 2.6: Logo di React Native

Nel 2012, Mark Zuckerberg, il celebre fondatore e CEO di Facebook, dichiarò che il più grande errore fatto dall'azienda fu quello di utilizzare un approccio web anziché quello nativo per realizzare la versione mobile dell'applicazione, promettendo che Facebook avrebbe presto migliorato l'esperienza degli utenti mobili [25]. Fu così che Facebook nel 2015 rilasciò il framework React Native, che oggi viene mantenuto e supportato anche da privati ed aziende di tutto il mondo, fra cui Callstack, Expo e Microsoft.

Il progetto ebbe ottimi risultati, tant'è che ben presto React Native si impose come uno dei framework multipiattaforma più utilizzati in assoluto, ed oggi vanta una delle community più attive e numerose.

2.3.1 Il funzionamento del framework

React Native nasce dalla fusione tra le tecnologie mobili e il framework React [26], da sempre uno dei più popolari in ambito di sviluppo web. Il framework consente di creare applicazioni del tutto assimilabili a quelle native mantenendo l'utilizzo di tecnologie web, e in particolare del linguaggio JavaScript, particolarmente diffuso ed amato dagli sviluppatori per la sua semplicità e facilità di apprendimento, e di CSS per modellare gli aspetti grafici dei vari componenti.

A livello logico, React Native basa il suo funzionamento su un bridge software che apporta alcune modifiche al codice JavaScript in modo da renderlo più vicino alle tecnologie native.

Il risultato di tale procedimento è un codice non ancora eseguibile sulle piattaforme mobili. Infatti, essendo ancora molto simile a JavaScript, esso necessita di essere interpretato, e solitamente ciò avviene all'interno di componenti inclusi in un browser. In ambito mobile, tuttavia, non esiste alcun

componente in grado di interpretare codice JavaScript senza appesantire in modo eccessivo l'esecuzione (come farebbe una WebView).

Per far fronte a tale situazione, React Native utilizza un motore JavaScript (JavaScript core) che effettua il passo decisivo per passare alle tecnologie native.

2.3.2 La realizzazione di un'applicazione React Native: i componenti

Il concetto fondamentale alla base di React Native, che consente allo sviluppatore di realizzare l'applicazione, è quello di componente.

Si tratta di un concetto ereditato da ReactJS, il framework di sviluppo web su cui React Native è basato: molti componenti utilizzabili in ReactJS sono compatibili con lo sviluppo nativo. È infatti possibile trasformarli tramite un bridge software in componenti nativi. Questi componenti sono quelli utilizzati da React Native per lo sviluppo di un'applicazione.

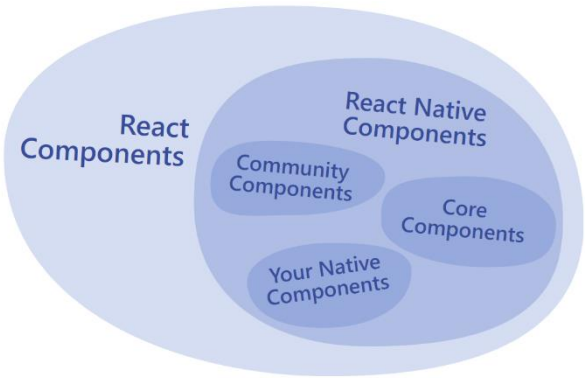

Figura 2.7: I componenti di React Native

Essi vengono rimodellati per poter essere utilizzati nello sviluppo di un'applicazione e vanno a costituire l'insieme dei Core Components di React Native. Si tratta di componenti base che modellano un singolo aspetto dell'interfaccia grafica e della logica dell'applicazione. Fra quelli più utilizzati e ricorrenti vi sono View, Text, Image, Button.

I componenti base del framework possono essere combinati tra di loro per realizzare componenti più complessi e articolati, per realizzare le specifiche esigenze dello sviluppatore.

Il programmatore React Native può inoltre decidere di appoggiarsi su componenti già pronti da utilizzare, creati dalla vasta community di React Native e disponibili online.

La figura 2.7 mostra come è possibile suddividere i componenti di React Native da un punto di vista puramente logico.

2.3.3 State e Props

Ci sono altri due concetti fondamentali che è bene comprendere per realizzare un'applicazione con React Native: State e Props. Attraverso essi è possibile gestire i componenti per tutto il loro ciclo di vita.

Quando si parla di Props si intende l'insieme delle proprietà di un componente (properties, da cui "props"). All'atto della creazione di un componente, esso riceve dal suo genitore l'insieme di proprietà che vengono impostate in fase di inizializzazione e sono immutabili per tutto il suo ciclo di vita. Ad esempio, se si considera il componente Image, sarà possibile impostare la props "source" per impostare l'URL della risorsa.

Diversi valori di props di uno stesso componente consentono di ottenere risultati e comportamenti differenti.

A differenza delle props, lo stato (State) di un componente è mutabile. Attraverso lo State è quindi possibile modificare il componente durante il suo ciclo di vita. Per farlo viene sfruttata la funzione setState(), che segnala la necessità di modificare l'elemento in questione. Ciò porterà al re-rendering del componente e di tutti i suoi figli in modo che l'interfaccia utente sia aggiornata di conseguenza. Attraverso lo State è quindi possibile gestire eventi e interazioni con l'utente.

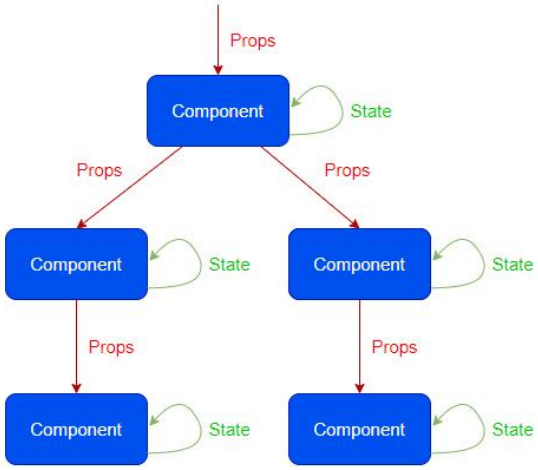

Figura 2.8: State e Props

La figura 2.8 mette in evidenza i concetti di state e props di un componente React Native.

2.3.4 React Native e l'alternativa Expo

Quando si comincia ad utilizzare React Native, è possibile decidere di utilizzare direttamente il framework completo, oppure approcciare lo sviluppo di applicazioni web native tramite Expo, suggerito anche nella documentazione ufficiale di React Native.

Si tratta di un'ottima scelta per sviluppatori alle prime armi, in quanto Expo rende molto semplice l'utilizzo di queste tecnologie, limitando però la possibilità di utilizzare funzionalità avanzate offerte da React Native.

Uno dei maggiori vantaggi dell'utilizzo di Expo è quello di poter testare direttamente l'applicazione durante lo sviluppo tramite il proprio dispositivo mobile, utilizzando un'apposita applicazione (Expo client), anziché dover passare dai classici ambienti di sviluppo mobile Android Studio e Xcode.

Tra le mancanze cui Expo non riesce a sopperire va segnalata in particolare l'impossibilità di integrare codice nativo al codice JavaScript, a differenza di quanto è possibile fare con React Native.

A causa di limitazioni di questo tipo, i programmatori più esperti e che desiderano avere accesso ad un'ampia gamma di funzionalità tendono sempre a preferire l'utilizzo di React Native rispetto ad Expo.

2.4 Flutter

Flutter è il framework per lo sviluppo di applicazioni multipiattaforma creato da Google. La figura 2.9 mostra il logo del framework.

Tra gli strumenti di sviluppo scelti, Flutter è quello più giovane e si sta pian piano affermando tra quelli preferiti dagli sviluppatori.

Figura 2.9: Logo di Flutter

Il progetto, inizialmente chiamato Sky, fu annunciato per la prima volta al Dart Developer Summit nel 2015. A partire da Sky, i successivi sviluppi portarono Google al rilascio di alcune versioni beta di Flutter fra il 2017 e il 2018, fino ad arrivare, il 4 dicembre 2018, al rilascio della prima versione stabile del framework.

Il progetto Flutter è open source e completamente gratuito, e consente agli sviluppatori di creare applicazioni multipiattaforma per iOS e Android di qualità e in tempi rapidi. Il framework viene continuamente migliorato, oltre che da Google stessa, dalla sua ampia community. Flutter viene inoltre utilizzato da aziende di grandi dimensioni, fra cui per esempio Alibaba.

2.4.1 Architettura del Framework

Flutter consente agli sviluppatori di creare applicazioni multipiattaforma utilizzando il linguaggio di programmazione Dart. Sviluppato da Google, si tratta di un linguaggio dinamico e object-oriented, molto simile a Java.

Il framework è costruito secondo un'architettura a livelli, come mostrato nella figura 2.10, in cui ogni strato è composto da librerie indipendenti che si appoggiano sul livello sottostante. Si tratta di un'architettura che ricorda quella proposta da Android.

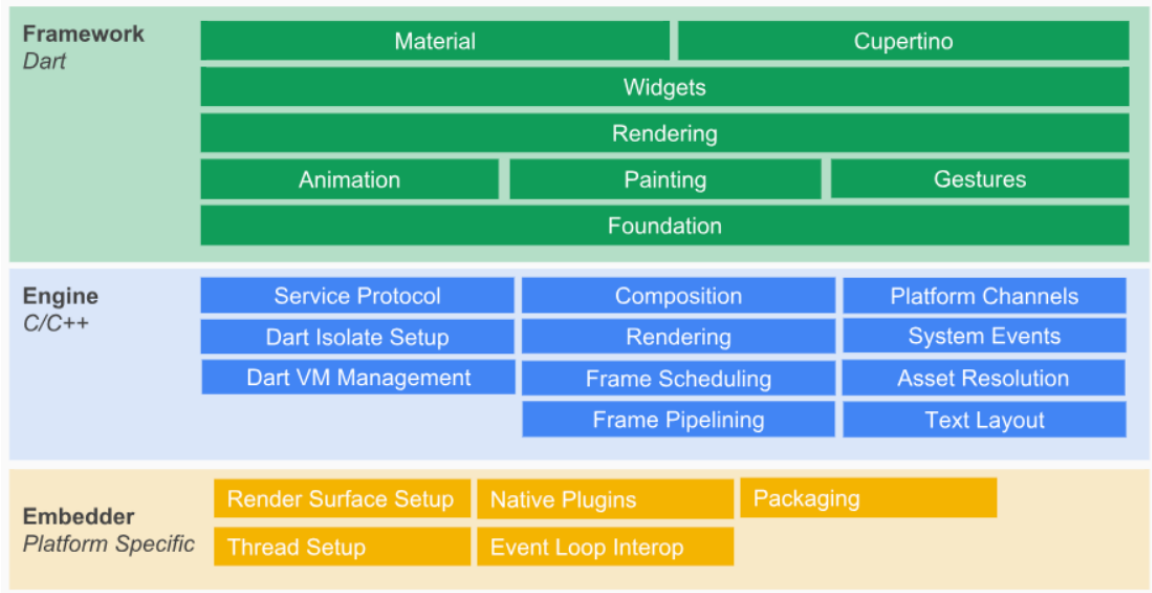

Figura 2.10: I tre strati dell'architettura di Flutter

Il livello inferiore della struttura del framework, denominato Embedder, è specifico della piattaforma. Al suo interno si posizionano librerie e componenti che forniscono un punto di accesso alle funzionalità native. È infatti questo livello che consente a Flutter di comunicare con il sistema operativo, di accedere a servizi di accessibilità ed input e molto altro ancora. Gli embedder specifici della piattaforma, per poter fornire tali servizi, si interfacciano al livello superiore dell'architettura, l'Engine, tramite API di basso livello scritte in un linguaggio che dipende dal sistema operativo, Java e C++ per Android, Objective-C e Objective-C++ per iOS.

All'interno di questo strato si trova inoltre una shell, specifica per ciascuna piattaforma, che fornisce l'accesso alle API native e che ospita la Dart VM, necessaria per l'esecuzione dell'applicazione.

Lo strato centrale, chiamato Engine, è il vero cuore dell'architettura del framework. Scritto in C++, esso comprende tutto ciò che è necessario per supportare una qualsiasi applicazione Flutter.

L'engine include infatti un numero veramente elevato di componenti di basso livello che rivestono un ruolo cruciale per il buon funzionamento del framework e per gestire le operazioni di base. Tra tutti i componenti che vanno a comporre l'engine di Flutter, sicuramente va citato il motore grafico Skia, responsabile del rendering.

Il livello di Engine viene esposto al programmatore tramite la libreria dart:ui, che implementa il codice C++ in classi Dart utilizzabili direttamente dallo sviluppatore.

Il livello più alto dell'architettura è quello denominato Framework. Si tratta dello strato più importante per gli sviluppatori, in quanto fornisce tutte le librerie da utilizzare per lo sviluppo di un'applicazione.

Fra esse vi sono componenti relativi a definizione di gesture, animazioni e soprattutto il layer Widget, che è senza dubbio il più importante per la creazione dell'app. Esso consente infatti di creare i widget, i componenti grafici dell'applicazione. Il programmatore può decidere di appoggiarsi a quelli già predisposti da Flutter, creati sulla base delle linee guida delle piattaforme e definiti nelle librerie Material per Android e Cupertino per iOS, oppure di creare dei widget personalizzati.

Andando a semplificare il tutto con una rappresentazione a più alto livello, si può dire che un'applicazione realizzata in Flutter è composta da widget, i quali possono rappresentare ad esempio contenitori, bottoni, testi, immagini. Tramite il motore grafico Skia, responsabile del rendering dell'applicazione, ciascun widget viene poi interpretato e rappresentato su una Canvas. Il risultato viene quindi mostrato all'utente dalla piattaforma, che si occupa di intercettare e gestire tutti gli eventi che scaturiscono dall'utilizzo dell'app. I componenti base coinvolti nell'implementazione e nell'esecuzione di un'applicazione Flutter e la loro interazione sono mostrati nella figura 2.11.

Figura 2.11: Funzionamento base di Flutter

2.4.2 La costruzione di un'applicazione Flutter: i widget

Con Flutter, la creazione dell'applicazione si riduce all'utilizzo dei Widget. Infatti, in un'app realizzata con il framework di Google, qualsiasi componente strutturale come un bottone o un menu, qualsiasi aspetto di layout, come margini o padding, o ancora il riconoscimento di gesture, viene definito come un widget.

La particolarità di questi elementi caratteristici di Flutter consiste nel fatto che attraverso essi è possibile costruire strutture gerarchiche all'interno delle quali ciascun widget che ne fa parte eredita varie proprietà dal widget padre. Spesso quindi si combinano tra loro vari widget anche piuttosto semplici per ottenere una struttura più elaborata.

I widget già predisposti da Flutter sono implementati in classi progettate secondo una struttura ad albero in cui la radice è la classe base Widget, che ha due figli che rappresentano le due tipologie fondamentali di widget: StatelessWidget e StatefulWidget.

I widget di tipo stateless sono caratterizzati da un tipo di layout statico, ovvero non è possibile modificare il loro aspetto a runtime. Essendo sprovvisti di uno stato, nessun evento o azione dell'utente può modificare il contenuto e l'aspetto di widget di questo tipo. Esempi di widget di tipo stateless possono essere Text, Container, Column.

I widget di tipo stateful sono invece quelli caratterizzati da un layout dinamico, cioè che prevedono uno stato, per cui è possibile modificarne il contenuto durante l'esecuzione, sulla base di eventi o di azioni dell'utente. Esempi di widget di tipo stateful possono essere Form e Scrollable.

A partire dai widget già pronti da utilizzare è naturalmente possibile creare componenti personalizzati, che a loro volta devono essere definiti come StatelessWidget o come StatefulWidget, in base al fatto che sia necessario modificarne il contenuto a runtime o meno.

2.4.3 Il processo di compilazione in Flutter

Rispetto agli altri framework per lo sviluppo multipiattaforma, Flutter si caratterizza per l'assenza del bridge, uno strato software intermedio che effettua operazioni di trasformazioni sul codice scritto dal programmatore. Al contrario, il codice Dart viene direttamente trasformato in codice eseguibile dalle piattaforme attraverso il processo di compilazione.

In fase di rilascio dell'applicazione, Flutter compila il codice C e C++ dell'Engine con l'aiuto di NDK (Native Development Kit) in Android e di LLVM (Low-Level Virtual Machine) in iOS. A questo punto il codice Dart viene compilato in modalità AOT (Ahead Of Time) e trasformato in codice nativo eseguibile dalla specifica piattaforma. Se la compilazione va a buon fine, viene così generato il file APK (Android) o IPA (iOS). Le operazioni di rendering dei widget, input/output, gestione di eventi vengono gestite in fase di esecuzione dell'applicazione.

Inoltre, Flutter, e più specificatamente il linguaggio Dart, consente anche di compilare il codice in modalità JIT (Just-In-Time). Questo tipo di compilazione è particolarmente utile durante la fase di sviluppo dell'applicazione, in quanto è possibile sfruttare la Dart VM per l'hot reload (ricarica a caldo) dell'applicazione e ridurre così i tempi necessari per compilare il codice una volta effettuate modifiche.

2.5 Similitudini e differenze tra i tre framework

Xamarin, Flutter e React Native sono tre framework che consentono di creare applicazioni multipiattaforma, eseguibili quindi su qualsiasi dispositivo indipendentemente dal sistema operativo. Essi rappresentano quindi tre alternative per lo sviluppatore per arrivare alla realizzazione di un'applicazione. Se da una parte vi sono degli aspetti in cui Xamarin, React Native e Flutter si assomigliano, dall'altra ciascun framework ha ovviamente le sue particolarità che lo differenziano dagli altri.

2.5.1 Aspetti in comune

I tre framework presentano alcune caratteristiche condivise:

- sono sviluppati da aziende leader nel settore – Xamarin è di proprietà di Microsoft, Flutter di Google e React Native di Facebook;
- open-source – i tre framework sono tutti open-source e ciò consente agli sviluppatori di avere a disposizione il codice sorgente per effettuare studi e modifiche condividendole con l'intera community;
- linguaggio dedicato – tutti e tre consentono di realizzare un'applicazione multipiattaforma utilizzando un linguaggio specifico. Xamarin usa C#, Flutter utilizza Dart, mentre con React Native si sviluppa in JavaScript;
- componenti standard – come la maggior parte dei framework di sviluppo, Xamarin, React Native e Flutter mettono a disposizione del programmatore una serie di componenti predefiniti e pronti da utilizzare;
- possibilità di scrivere codice nativo – pur essendo basati su linguaggi specifici, Xamarin, React Native e Flutter consentono al programmatore di scrivere codice nativo per realizzare specifiche funzionalità.

Si deve inoltre sottolineare che tutti e tre i framework, essendo stati ideati per la creazione di applicazioni cross-platform, consentono di raggiungere con un unico codice tutte le piattaforme. La stragrande maggioranza del codice C#, JavaScript e Dart sarà infatti compatibile sia per iOS che per Android.

2.5.2 Differenze tra i framework

Accanto a tutti questi aspetti comuni, analizzando le caratteristiche di Xamarin, React Native e Flutter è possibile individuare alcune differenze.

2.5.2.1 Architettura: il diverso approccio di Flutter

Un primo aspetto in cui uno dei tre framework adotta una soluzione differente rispetto agli altri è quello dell'architettura.

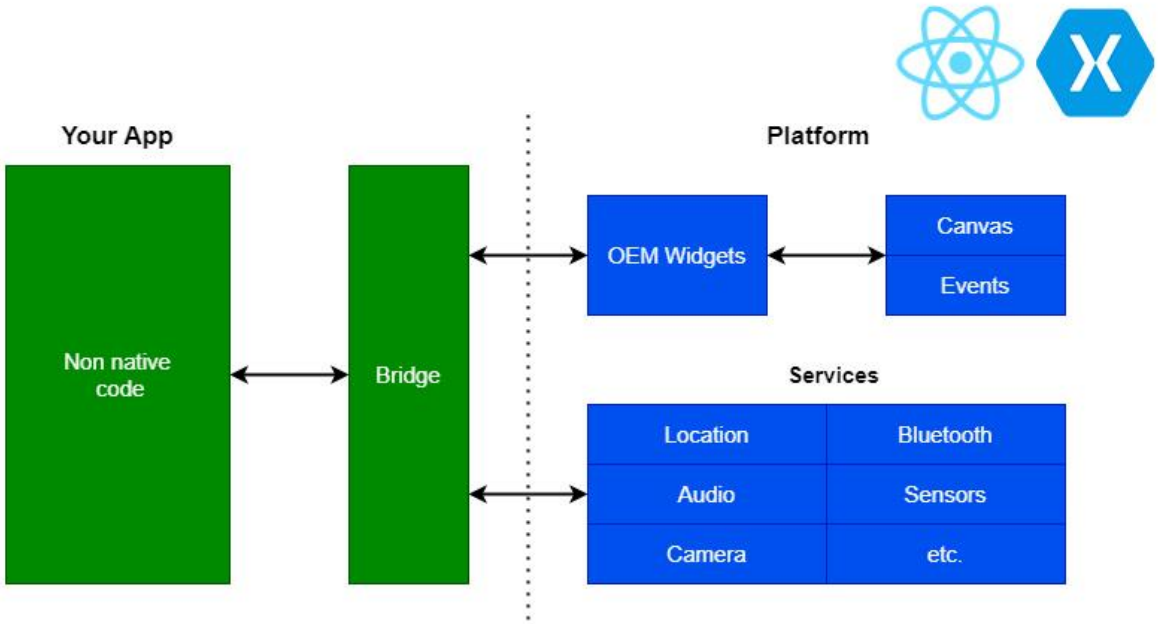

Figura 2.12: Architettura di Xamarin e React Native

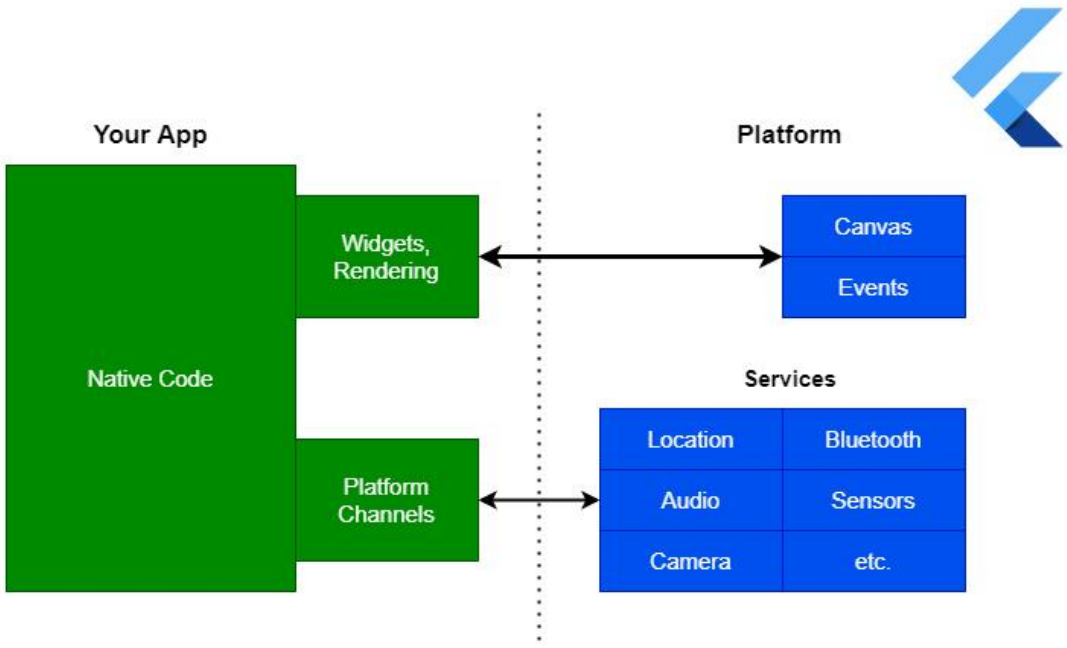

Figura 2.13: Architettura di Flutter

Xamarin e React Native hanno una struttura praticamente equivalente, che prevede la presenza del bridge software tra il codice implementato e i componenti della specifica piattaforma. I componenti realizzati con il linguaggio specifico del framework vengono trasformati in widget OEM (Original Equipment Manufacturer), cioè in componenti nativi.

L'architettura di Xamarin e React Native, schematizzata nella figura 2.12, prevede che anche il codice scritto per l'utilizzo dei servizi del dispositivo, quali ad esempio audio, sensori e fotocamera, passi attraverso il bridge per poter essere correttamente eseguito.

Flutter invece utilizza un approccio decisamente diverso. Il framework di Google infatti non prevede la presenza del bridge né dei widget OEM, ma utilizza dei componenti (i widget di Flutter) direttamente trasformabili in codice eseguibile. Anche per interfacciarsi ai servizi del dispositivo, quali ad esempio Bluetooth e GPS, non c'è lo strato di bridge ma si utilizzano gli appositi strumenti denominati Platform Channels. L'architettura alternativa proposta da Flutter è schematizzata nella figura 2.13.

2.5.2.2 Prestazioni, hot reload e dimensioni dell'applicazione

Tutti e tre i framework utilizzati garantiscono prestazioni naturalmente inferiori rispetto a quelle delle applicazioni native, ma comunque si tratta di performance decisamente buone che fanno sì che Xamarin, React Native e Flutter rappresentino ottime soluzioni per lo sviluppo di applicazioni mobili.

La diversa architettura di Flutter rispetto agli altri due framework consente di fare ulteriori considerazioni. L'assenza dello strato intermedio di bridge garantisce infatti prestazioni migliori durante l'esecuzione dell'applicazione. Se infatti React Native adotta l'approccio dell'interpretazione tramite il motore JavaScript e Xamarin compila il codice C# in un codice intermedio che viene poi ritrasformato in fase di esecuzione, con Flutter il codice Dart viene compilato in modalità AOT direttamente in codice binario eseguibile. Ciò consente di incrementare le prestazioni in quanto il formato precompilato può essere eseguito in modo diretto, effettuando calcoli complessi e operazioni di rendering in maniera più efficiente.

Tutti e tre i framework supportano la possibilità di effettuare il cosiddetto "hot reload" (chiamato "fast refresh" in React Native), ovvero la possibilità di osservare in modo praticamente istantaneo le modifiche apportate all'applicazione in fase di sviluppo. Ciò consente di evitare sprechi di tempo per ricompilare tutto il codice e riavviare l'applicazione ogni volta che viene effettuata una singola modifica. In tale contesto, l'unica limitazione che si riscontra utilizzando Xamarin è che l'hot reload è supportato solamente per la componente grafica dell'applicazione (codice XAML), mentre se si apportano modifiche alla logica applicativa (codice C#) sarà necessario riavviare l'applicazione per poterla testare.

In termini puramente teorici, si può quindi considerare Flutter leggermente migliore rispetto a React Native e Xamarin per quanto riguarda le prestazioni dell'applicazione finale. Inoltre, l'impossibilità di eseguire l'hot reload per la parte logica fa sì che Xamarin risulti leggermente sfavorito rispetto agli altri due framework in termini di velocità nello sviluppo dell'applicazione. Si tratta comunque di considerazioni che dipendono in gran parte dall'applicazione realizzata e in particolare dalle funzionalità implementate, dall'utilizzo di librerie e file di terze parti, dallo sfruttamento di servizi esterni e così via.

Se la soluzione architetturale di Flutter comporta un leggero vantaggio nelle prestazioni rispetto a Xamarin e React Native, dall'altro lato ciò introduce anche uno svantaggio che riguarda le dimen-

sioni dell'applicazione. Infatti, oltre al codice dell'applicazione, essa conterrà anche i widget e i componenti per effettuare l'accesso ai servizi della piattaforma, mentre in Xamarin e React Native queste operazioni vengono svolte dal bridge software intermedio che risulta decisamente più leggero.

Tra i tre framework, quello che consente di realizzare in genere le applicazioni più leggere dal punto di vista della memoria occupata nel dispositivo è senza dubbio React Native. Analogamente a quanto detto per le prestazioni, anche le considerazioni sulle dimensioni dipendono in ogni caso dalla realizzazione dell'applicazione, e quindi ancora una volta dalle funzionalità, dall'utilizzo di librerie esterne, dallo sfruttamento di servizi di terze parti e così via.

2.5.2.3 Grafica ed esperienza utente

A differenza di Xamarin, Flutter e React Native dispongono di un meccanismo che consente di gestire il rendering dei componenti in modo ottimale, rendendo così l'applicazione più performante in caso di una grafica complessa.

In Flutter, il rendering è affidato al motore grafico Skia: i widget vengono organizzati secondo una struttura ad albero e, apportata una modifica ad un componente, viene eseguito solamente il rendering di esso e dei suoi figli, in modo da rendere efficiente la visualizzazione dei contenuti.

In React Native si sfrutta un meccanismo simile: anche in questo caso, tramite la funzionalità del Virtual DOM, viene creato un albero dei componenti e una volta effettuata la modifica, verrà effettuato il re-rendering di quel componente e di tutti i suoi figli.

L'assenza di un analogo meccanismo in Xamarin, insieme alla necessità di dover mantenere separata la parte grafica dalla parte logica dell'applicazione, rende il framework di Microsoft un'ottima soluzione per applicazioni con interfaccia utente relativamente semplice, ma non particolarmente adatto in tutti quei casi in cui l'applicazione da realizzare presenta una grafica complessa e particolarmente pesante.

Dall'altra parte, Xamarin è fra i tre il miglior framework per quanto riguarda il rispetto delle linee guida delle singole piattaforme per quel che riguarda l'interfaccia e l'esperienza utente, che con Xamarin risultano pressoché identiche a quelle che si ottengono sviluppando in nativo. Se si programma con approccio Xamarin.Forms, in particolare, si avranno a disposizione elementi dell'interfaccia utente standard che verranno associati in fase di compilazione a quelli della specifica piattaforma. Se invece si sceglie l'approccio classico, quello di Xamarin.Android e Xamarin.iOS si andrà a realizzare l'interfaccia utente direttamente con le tecnologie native.

3. Applicazione MyMusic

Utilizzando i tre diversi framework multipiattaforma (Flutter, Xamarin e React Native), è stata realizzata un'applicazione mobile, chiamata "MyMusic". Quest'ultima è stata implementata con le stesse funzionalità, con lo scopo di far emergere le differenze tra i tre strumenti di sviluppo utilizzati.

3.1 Il dominio dell'applicazione

MyMusic, come si evince dal nome, è un prodotto il cui campo applicativo è quello dell'ambito musicale. Gli utenti che utilizzano l'applicazione potranno ricavare informazioni su artisti musicali ed ascoltare canzoni, di cui saranno visibili il video e il testo. Inoltre, registrando un account MyMusic, gli utenti potranno salvare artisti e canzoni nei propri preferiti, in modo da poter consultare le relative informazioni più rapidamente. Inoltre, essendo i dati dei preferiti salvati in un database esterno e non nella memoria interna, sarà possibile per ciascun utente accedere ai propri dati su qualsiasi dispositivo.

3.2 L'architettura dell'applicazione

L'applicazione, per poter implementare le varie funzionalità, si appoggia su risorse esterne per ricercare e memorizzare le informazioni, come mostrato nella figura 3.1.

In particolare, i dati relativi a canzoni e artisti vengono raccolti sfruttando le API della nota piattaforma Genius [65], mentre la ricerca dei testi delle canzoni avviene tramite le API di lyrics.ovh [66].

Per registrare gli account degli utenti, per salvare i loro preferiti e per suggerire artisti e canzoni da ascoltare viene utilizzata la piattaforma Firebase [67].

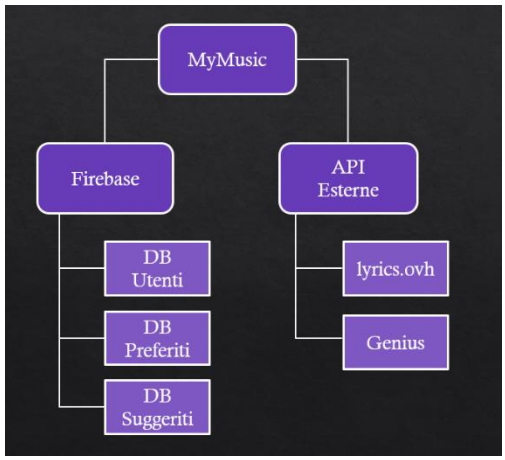

Figura 3.1

3.2.1. API Esterne – Genius e lyrics.ovh

La maggior parte delle informazioni utilizzate per realizzare l'applicazione è ricavata tramite le API di Genius.

La piattaforma mette a disposizione un'API di ricerca globale "/search" che fornisce come risultato una lista di dieci canzoni sulla base dell'input fornito. Sfruttando questa funzionalità, è possibile ottenere l'ID di una determinata canzone e del relativo artista. Una volta ottenuti gli ID, si è in grado di accedere ad informazioni molto più dettagliate utilizzando le API "/artists/:id" e "/songs/:id".

Quest'ultime mettono a disposizione una quantità veramente ampia di dati. Ai fini dell'applicazione si è deciso di limitarsi ad utilizzarne una piccola parte.

Per quanto riguarda gli artisti, oltre al nome, sono stati sfruttati i campi "description" per ottenere la biografia e "header_image_url" per ricavare un'immagine.

Per le canzoni, oltre al titolo e al nome dell'artista, è stato utilizzato il campo "media" per risalire all'indirizzo URL del video musicale caricato su YouTube.

Per poter utilizzare il servizio, Genius richiede allo sviluppatore di autenticarsi utilizzando un Bearer Token, fornito gratuitamente dalla piattaforma al momento della registrazione dell'account.

Oltre alle informazioni ricavate dalle API di Genius, per quanto riguarda le canzoni è stato inserito anche il testo, che può essere consultato durante l'ascolto e viene ricavato sfruttando lyrics.ovh. Si tratta di un sito che, attraverso un'API molto semplice, consente di ottenere il testo di una canzone. È sufficiente specificare nome dell'artista e titolo della canzone per ottenere in risposta un oggetto composto da un unico campo che contiene direttamente il testo.

3.2.2 Firebase

La piattaforma Firebase, sviluppata e gestita da Google, è un servizio di "backend as a service" (BaaS), che fornisce agli sviluppatori di applicazioni web e mobili un'ampia gamma di funzionalità.

L'applicazione MyMusic utilizza il servizio di registrazione account e di autenticazione degli utenti. Firebase consente infatti di registrare un account tramite una e-mail e una password e consente inoltre di effettuare operazioni quali login e modifica password.

Inoltre, viene sfruttata la possibilità di gestire veri e propri database (Cloud Firestore o Realtime Database) per memorizzare informazioni aggiuntive sugli utenti e le loro canzoni e artisti preferiti. Queste funzionalità vengono utilizzate anche per predisporre un database contenente le canzoni e gli artisti suggeriti dal team MyMusic.

3.3 Funzionalità e Layout dell'Applicazione

All'avvio dell'applicazione viene mostrata una pagina iniziale (figura 3.2) che presenta all'utente le sezioni delle canzoni e degli artisti suggeriti dal team MyMusic. Si tratta di una serie di canzoni ed artisti pronti da visualizzare, in modo che l'utente possa subito capire quelle che sono le funzionalità offerte dall'applicazione.

Con la barra di navigazione inferiore sarà poi possibile spostarsi nelle altre pagine, che consentono all'utente di cercare canzoni ed artisti specifici e gestire il proprio profilo.

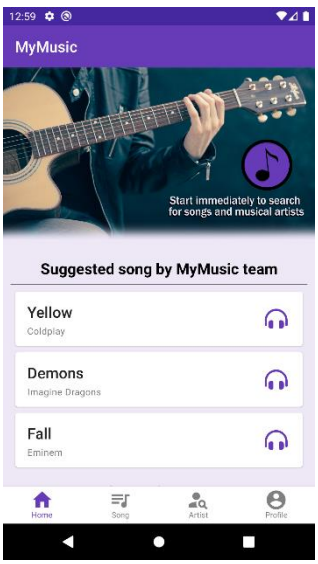

Figura 3.2

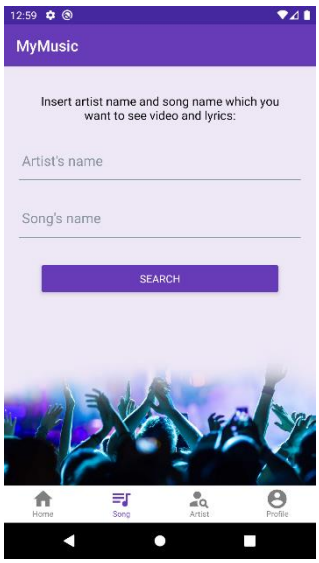

Figura 3.3

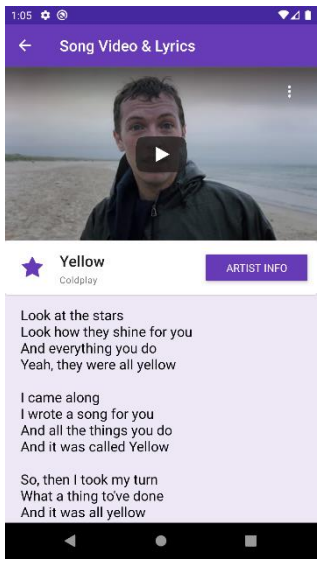

Figura 3.4

La funzionalità di ricerca canzone (figura 3.3) consente all'utente di specificare una canzone, inserendo il titolo e il nome dell'artista. Se la ricerca va a buon fine, verrà mostrata all'utente la

pagina dei dettagli della canzone (figura 3.4), in cui verranno mostrati il video e il testo, oltre ad un bottone che rimanda alla pagina dettagli artista (figura 3.6).

Inoltre, se l'utente si è precedentemente loggato, sarà possibile aggiungere (o rimuovere) la canzone dai propri preferiti utilizzando l'apposita icona.

Analogamente, la funzionalità di ricerca artista (figura 3.5) permette di inserire il nome di un'artista: se la ricerca va a buon fine, verrà mostrata la pagina dei dettagli dell'artista (figura 3.6), contenente un'immagine, la biografia, e una lista delle sue principali canzoni (figura 3.7).

Anche in questo caso, se l'utente ha effettuato il login, sarà possibile aggiungere (o rimuovere) l'artista dai propri preferiti, servendosi dell'apposita icona.

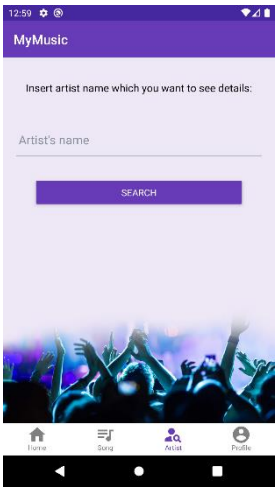

Figura 3.5

Figura 3.6

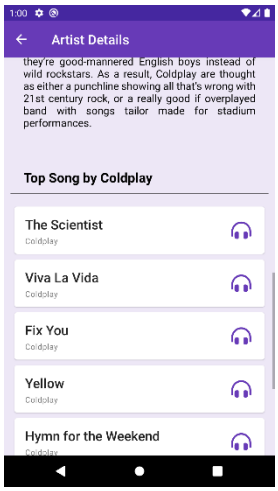

Figura 3.7

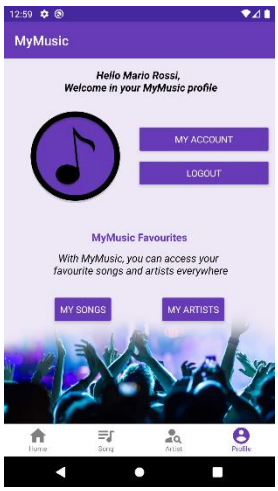

Figura 3.8

La schermata relativa al profilo consente all'utente non loggato di effettuare il login, o eventualmente di registrarsi nel caso in cui non possieda un account MyMusic. Una volta effettuata l'autenticazione, viene mostrata all'utente la propria area riservata (figura 3.8).

L'utente potrà accedere ai propri preferiti, utilizzando gli appositi bottoni "My Songs" e "My Artists", che rimandano all'elenco delle canzoni e degli artisti preferiti dell'utente, da cui sarà possibile visualizzarli oppure rimuoverli dai preferiti.

4. Implementazione di MyMusic in React Native

Dopo aver descritto e confrontato Xamarin, React Native e Flutter, e dopo aver introdotto l'applicazione MyMusic, si andrà ora a parlare di come essa è stata realizzata.

Ciascun componente del gruppo di lavoro ha utilizzato uno dei tre framework scelti per l'implementazione di MyMusic, realizzando le stesse funzionalità e cercando di mantenere quanto più possibile comune l'interfaccia e l'esperienza utente, in modo da far emergere eventuali problematiche riscontrate durante l'implementazione e derivanti dall'utilizzo di uno specifico framework piuttosto che di un altro.

Nello specifico, chi scrive il presente elaborato ha realizzato l'applicazione utilizzando React Native e di seguito descrive l'implementazione con il framework di Facebook Inc..

4.1 Scheletro dell'applicazione

Lo scheletro dell'applicazione è strutturato in due file: App.js e TabNavigator.js.

Il primo citato contiene la completa gestione della navigazione dell'applicazione grazie all'implementazione della libreria "React Navigation" che consente di costruire la comunicazione tra le varie pagine inserendo anche la top bar e modificandone lo stile a proprio piacimento. Ciò avviene grazie all'aggiunta di un componente che è buona norma chiamare semplicemente Stack, se è l'unico componente utilizzato per la gestione della navigazione. Il gruppo di navigazione costruito comprende lo scheletro delle pagine della bottom navigation e cinque pagine differenti, tra cui quella che comprende il testo e il video della canzone, la pagina che contiene i dettagli dell'artista, quella che consente di modificare la password del proprio account e le due pagine che racchiudono la raccolta delle canzoni preferite e degli artisti preferiti dell'utente.

Inoltre tramite il componente StatusBar è possibile cambiare il colore della status bar durante l'utilizzo dell'applicazione. Sono state seguite le linee guida dei due sistemi operativi, infatti per Android è stata utilizzata una tonalità più scura del colore della top bar, mentre per iOS è stato lasciato lo stesso colore della top bar.

Per quanto riguarda il file TabNavigator.js gestisce, tramite la creazione di una componente denominata Tab, le proprietà e lo stile del bottom navigation. Quest'ultimo comprende quattro schermate differenti: la home, il profilo, la ricerca della canzone e quella dell'artista.

Le icone utilizzate nell'applicazione sono state scelte importando la libreria esterna "Vector Icons" che ne comprende più di tremila differenti e in particolare sono state usate quelle appartenenti ai sottogruppi "MaterialIcons" creato da Google, "FontAwesome5" creato da Fonticons e "Feather" creato da Cole Bemis & Contributors .

Inoltre sono state importate altre due librerie esterne per ottenere una buona interfaccia grafica: React Native Paper per le card e React Native Elements per gli input text.

Per modificare i layout delle varie pagine e gli stili dei vari componenti, in React Native viene utilizzato il componente StyleSheet, un'astrazione simile ai StyleSheet di CSS. Questo viene posto alla fine di ogni file e permette di raccogliere in un unico blocco tutti gli stili di quella determinata pagina, in questo modo il codice risulterà più pulito e ordinato.

4.2 Schermata principale

Classi coinvolte nella schermata home		
Classe	Descrizione	Servizi utilizzati
Home	Mostra le canzoni e gli artisti suggeriti	Firebase Firestore Database

La schermata principale, nonché la Home dell'applicazione, è impostata per essere la prima pagina visibile subito dopo l'esecuzione dell'applicazione. Essa è costituita da un'immagine introduttiva, implementata con la componente Image, e due elenchi che comprendono le canzoni e gli artisti suggeriti dal team MyMusic. Quest'ultimo contenuto è stato inserito tramite il componente FlatList caricando, con useEffect, due database presenti su firestore e salvati nello state tramite useState. Ogni elemento è all'interno di una card con la presenza del titolo della canzone, del nome dell'artista e di un'icona a destra.

L'icona, rappresentante un paio di cuffie per le canzoni e un foglio per gli artisti, è cliccabile in modo tale da accedere, tramite un push di React Navigation, ai dettagli della canzone o dell'artista suggerito.

4.3 Ricerca canzone e pagina dettagli canzone

Classi coinvolte nella schermata ricerca canzone e dettagli canzone		
Classe	Descrizione	Servizi utilizzati
SongSearch	Permette la ricerca di una canzone	-
SongScreen	Mostra il video e il testo della canzone cercata	LyricsOVH, Genius

La schermata per avviare la ricerca di una canzone, ottenendone il video e il testo, appartiene alla raccolta della bottom tab navigator. Questa pagina, oltre a contenere un titolo aggiunto con la componente Text e un'immagine in fondo aggiunta con la componente Image, racchiude due input text che permettono di inserire il nome dell'artista e della canzone e un bottone che ne consente la ricerca. In ognuno dei due input text è stato aggiunto un placeholder che suggerisce all'utente cosa dovrà inserire nelle caselle di testo, un messaggio di errore che appare al momento della ricerca se la casella di testo risultasse vuota e un'icona che permette di svuotarlo se riempito.

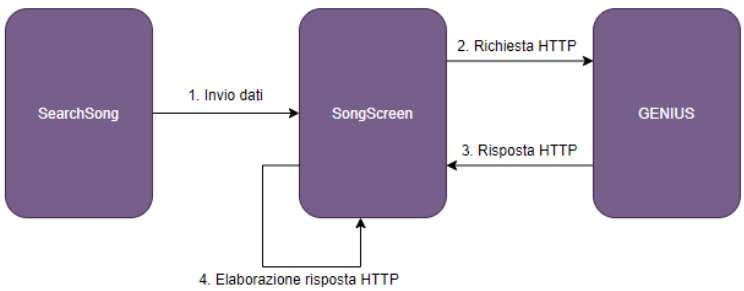

Figura 4.1: la logica di funzionamento della ricerca canzone, nel caso in cui la ricerca vada a buon fine

Il bottone di ricerca è stato implementato tramite il componente Button. Esso, appena viene premuto, fa il controllo sugli input text, controllando che non siano vuoti e cambiando lo stato di validità del testo tramite useState, altrimenti effettua la ricerca tramite React Navigation con parametri il nome della pagina successiva salvato sullo stack di navigazione e le due stringhe inserite negli input text.

Se la ricerca va a buon fine verrà visionata la pagina contenente i dettagli della canzone, in caso contrario uscirà un messaggio di errore e sarà possibile tornare indietro tramite la freccia verso sinistra presente sulla top bar. La pagina caricata correttamente prevede la visualizzazione del video e delle lyrics della canzone e la possibilità di accedere ai dettagli dell'artista.

Appena la pagina viene caricata, viene effettuato un controllo per assicurarsi che l'utente sia loggato, in modo tale da far comparire l'icona per il salvataggio nei preferiti. Ciò è possibile tramite la funzione useEffect dove all'interno vengono eseguiti dei metodi dell'autenticazione di firebase, infatti oltre a controllare se l'utente è loggato, effettua un controllo anche nel database di firestore per sapere se la canzone cercata risulta già nella raccolta dei preferiti di quel determinato utente.

In seguito vengono caricati tutti i componenti appartenenti alla pagina, dati pescati dalle rispettive API utilizzate: lyricsOVH e Genius. Il funzionamento è descritto nella figura 4.1. Tutto ciò avviene all'interno di un useEffect.

Per il testo, scrollabile grazie alla componente ScrollView in cui si trova, viene effettuata una ricerca tramite fetch inserendo l'URL dell'API lyricsOVH contenete il nome dell'artista e il nome della canzone. Se la ricerca va a buon fine verrà salvato il dato JSON tramite useState con delle modifiche in modo tale da avere il testo a schermo ripulito senza spazi inutili, altrimenti verrà eseguito il catch che fa apparire un alert sottolineando l'errore.

Per il resto delle informazioni viene effettuata una ricerca concatenata di fetch utilizzando l'API di Genius. La prima ricerca serve a salvare, partendo dai dati contenuti nel JSON e tramite useState, il nome dell'artista e della canzone, mentre la seconda ricerca serve a salvare l'ID del video YouTube della canzone. In tutti e due i casi, il catch porterà ad un cambio di stato che consentirà la visualizzazione del messaggio di errore al momento della ricerca.

Avendo tutti i dati disponibili per caricare la pagina in modo corretto e con i propri dettagli, la rotellina di caricamento visualizzata tramite la componente ActivityIndicator scomparirà grazie al cambio di stato che avviene nei finally dei fetch. Il video di YouTube sarà visualizzato tramite la libreria esterna "React Native YouTube iFrame" importata nel progetto. La card contiene, oltre al nome della canzone e al nome dell'artista, due componenti: LeftCardContent e RightCardContent. La prima inserisce un'icona stella che permette all'utente registrato e loggato di salvare o rimuovere dai preferiti la pagina in questione; mentre la seconda inserisce un bottone che permette all'utente di visualizzare i dettagli dell'artista tramite un push sullo stack della navigazione mostrando una nuova pagina.

4.4 Ricerca artista e pagina dettagli artista

Classi coinvolte nella schermata ricerca canzone e dettagli canzone		
Classe	Descrizione	Servizi utilizzati
ArtistSearch	Permette la ricerca di un artista	-
ArtistScreen	Mostra i dettagli dell'artista cercato	Genius

La schermata per avviare la ricerca di un'artista, ottenendone una biografia e una raccolta di canzoni, appartiene alla raccolta della bottom tab navigator. Questa pagina, oltre a contenere un titolo aggiunto con la componente Text e un'immagine in fondo aggiunta con la componente Image, racchiude un input text che permette di inserire il nome dell'artista e un bottone che ne consente la ricerca.

Nell'input text è stato aggiunto un placeholder che suggerisce all'utente cosa dovrà inserire nelle caselle di testo, un messaggio di errore che appare al momento della ricerca se l'input text risultasse vuoto e un'icona che permette di svuotare l'input text se riempito.

Il bottone di ricerca è stato implementato tramite il componente Button. Esso, appena viene premuto, fa il controllo sull'input text, controllando che non sia vuoto e cambiando lo stato di validità del testo tramite useState, altrimenti effettua la ricerca tramite React Navigation con parametri il nome della pagina successiva salvato sullo stack di navigazione e la stringa inserita negll'input text.

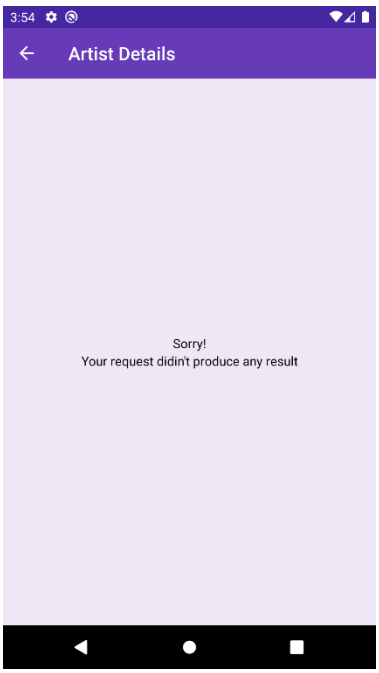

Figura 4.2: Messaggio di errore in caso di artista non trovato

Se la ricerca va a buon fine verrà visionata la pagina contenente i dettagli dell'artista, in caso contrario uscirà un messaggio di errore e sarà possibile tornare indietro tramite la freccia verso sinistra presente sulla top bar, come mostrato nella figura 4.2. La pagina caricata correttamente prevede la visualizzazione della biografia dell'artista e una sequenza di canzoni appartenenti a quest'ultimo.

Appena la pagina viene caricata, viene effettuato un controllo per assicurarsi che l'utente sia loggato, in modo tale da far comparire l'icona per il salvataggio nei preferiti, possibile tramite useEffect. All'interno di questa funzione vengono chiamati dei metodi dell'autenticazione di firebase, infatti oltre a controllare se l'utente è loggato, effettua un controllo anche nel database di firestore per sapere se l'artista cercato risulta già nella raccolta dei preferiti di quel determinato utente.

In seguito vengono caricati tutti i componenti appartenenti alla pagina, dati pescati dall'API Genius; tutto ciò avviene all'interno di un useEffect.

Per trovare le informazioni utili si utilizza una concatenazione di fetch. Nel primo si ottiene il dato JSON, utilizzato per salvare tramite useState sia l'array con all'interno tutte le canzoni da inserire nella sezione Top Song, sia il nome, l'immagine e l'id dell'artista in questione. Nel secondo viene salvata la biografia dell'artista. Nel database di Genius essa è spezzettata in sezioni per i vari link presenti in un testo di una pagina web, per questo è risultato necessario concatenare le varie parti tramite una serie di controlli e cicli salvando solo le stringhe che facevano parte di determinati tag.

Avendo tutti i dati disponibili per caricare la pagina in modo corretto e con i propri dettagli, la rotellina di caricamento visualizzata tramite la componente ActivityIndicator scomparirà grazie al cambio di stato che avviene nei finally dei fetch. Quindi comparirà la pagina completa implementata tramite la componente FlatList. Si è deciso di utilizzare questa componente per aggiungere senza particolari problemi tutte le canzoni presenti nell'array salvato in precedenza. Infatti, tramite la props ListHeaderComponent è stato aggiunto il nome dell'artista, la stella per la funzione "preferiti" e la biografia completa, mentre in seguito l'elenco delle top song, in cui ogni canzone ha il proprio id e la propria card. Ogni card ha un'icona a destra che permette di accedere alla pagina della canzone in cui è possibile visionare video e testo. Ciò avviene grazie al push sullo stack della navigazione, mostrando una nuova pagina.

4.5 Profilo e preferiti

Classi coinvolte nella schermata profilo		
Classe	Descrizione	Servizi utilizzati
Profile	Comprende le tre pagine: login, registrazione e profilo	Firebase Authentication
ManageProfile	Permette di modificare la password di un account	Firebase Authentication
FavoriteSongs	Mostra le canzoni preferite dell'utente loggato	Firebase Authentication, Firebase Firestore Database
FavoriteArtists	Mostra gli artisti preferiti dell'utente loggato	Firebase Authentication, Firebase Firestore Database

L'ultima pagina dell'applicazione appartenente alla bottom tab è quella che fa riferimento al profilo di un utente. Essa ha tre differenti schermate: quella utile per accedere al profilo, quella che consente la registrazione di un nuovo account e quella mostrata subito dopo aver effettuato il login. Per capire quale schermata visualizzare, viene effettuato un controllo tramite un metodo di Firebase Authentication in modo tale da sapere se un utente è già loggato e salvandolo nel caso in uno state tramite useState. La logica alla base della costruzione del contenuto della pagina relativa all'area dell'autenticazione o del profilo utente è descritta nella figura 4.3.

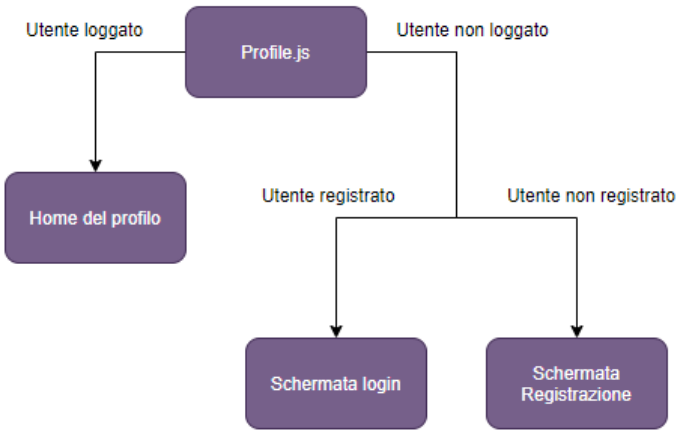

Figura 4.3: la logica di costruzione del contenuto della schermata profilo

Se l'utente non dovesse risultare loggato, l'applicazione mostrerebbe la pagina che consente il login tramite due input utili per l'immissione dell'email e della password appartenenti all'account. Il login avviene tramite un bottone implementato con la componente Button e un metodo di Firebase Authentication, effettuando il controllo sui campi vuoti e sull'esistenza dell'email e/o della password.

Se l'utente volesse invece effettuare la registrazione di un nuovo account, basterebbe premere il bottone in fondo alla pagina per cambiare la schermata in un form di registrazione. L'utente può tornare alla schermata di login premendo un nuovo bottone in qualsiasi momento. Per completare la registrazione bisognerà occupare tutti gli input text, Full Name, Email, Password e Confirm Password, e premere il bottone di registrazione. Vengono effettuati anche i controlli sull'input in modo tale da non lasciare campi vuoti, considerare solo email valide e confrontare i due campi delle password.

Dopo aver effettuato la registrazione o il login, la schermata sarà quella del profilo vero e proprio in cui il layout è composto da un componente Text che dà il benvenuto, da un componente Image che raffigura il logo dell'applicazione, due bottoni, My Account e Logout, che consentono rispettivamente di accede ad una nuova pagina per cambiare la password dell'account e di effettuare il logout passando alla schermata di login, ambedue processi possibili tramite due metodi di Firebase Authentication. In seguito si trova la sezione preferiti, in cui è possibile accedere alle due pagine che raccolgono le canzoni, per un bottone, e gli artisti, per l'altro bottone.

Le due pagine dei preferiti sono state implementate in modo completamente analogo, è cambiato solo il database a cui viene fatto riferimento; infatti per uno bisogna considerare il database degli artisti, per l'altro quello delle canzoni. All'interno di queste pagine viene utilizzato il componente FlatList, in modo da ottenere una lista, e il componente TouchableHighlight con all'interno una card da poter premere per accedere alla pagina della determinata canzone o del determinato artista. All'interno della funzione useEffect viene sfogliata la raccolta delle canzoni o degli artisti inseriti nei preferiti in modo da salvarla e aggiungere nella componente FlatList. Verrà salvato il nome dell'artista, il nome della canzone (solo per la pagina delle canzoni preferite) e l'id, tramite lo state. Inoltre sarà possibile rimuovere un elemento tra i preferiti tramite l'icona del cestino aggiunta a destra di ogni card, eliminando immediatamente l'artista o la canzone dal database e quindi la card dalla lista.

5. Confronto fra le varie implementazioni e problematiche riscontrate

Una volta implementata l'applicazione con ciascuno dei tre framework, i tre progetti sono stati messi a confronto e sono sorte alcune interessanti considerazioni. Nella maggior parte dei casi, non sono state riscontrate evidenti differenze derivanti dall'utilizzo di un framework piuttosto che di un altro, ma nella realizzazione di specifiche funzionalità sono sorte alcune criticità che sono state gestite in maniera differente.

5.1. Funzionalità comuni e interfacce molto simili

Pur avendo sviluppato MyMusic utilizzando tre framework diversi, in ciascuno dei tre progetti si è riusciti a realizzare tutte le funzionalità che erano state stabilite in fase di progettazione dell'applicazione.

Si è cercato inoltre per quanto più possibile di mantenere uguali le interfacce utente, aldilà delle caratteristiche grafiche dei componenti di ciascun framework. Infatti, sia Xamarin, sia React Native, sia Flutter mettono a disposizione una serie di componenti che hanno consentito di realizzare l'applicazione in modo soddisfacente, sia dal punto di vista del layout sia dal punto di vista della logica applicativa.

La figura 5.1 mostra il layout grafico delle pagine del profilo realizzate con i tre framework. Come si può vedere, salvo aspetti grafici davvero marginali, le tre interfacce risultano praticamente equivalenti.

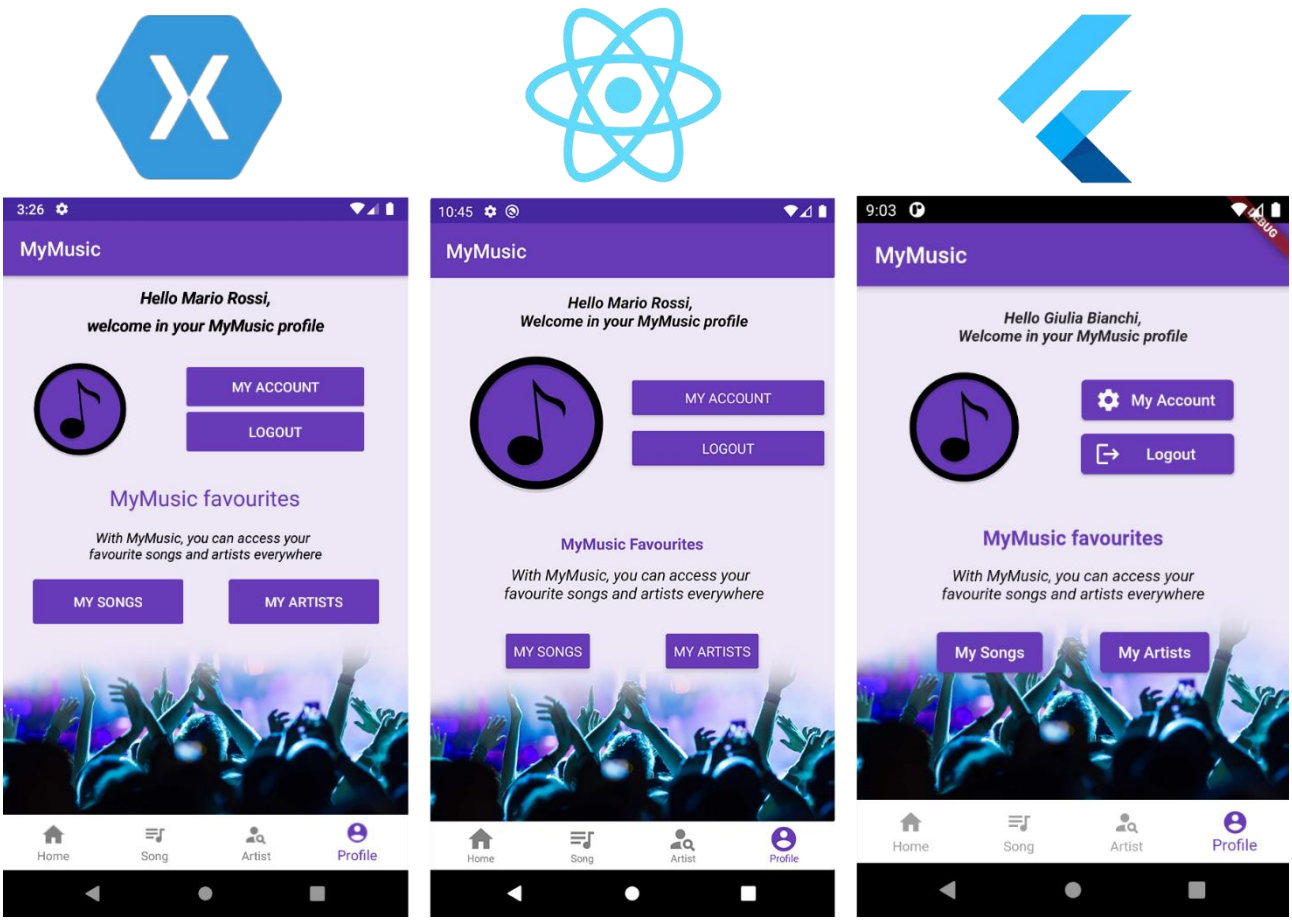

Figura 5.1: Le tre interfacce della pagina del profilo con Xamarin, React Native e Flutter

5.2 Problematiche e differenze

In alcune situazioni, tuttavia, sono venuti alla luce alcuni problemi derivanti dall'utilizzo di un framework piuttosto che di un altro. In questi casi è stato necessario apportare alcuni accorgimenti a seconda del framework utilizzato.

5.2.1 Navigazione con React Native

Se l'utilizzo di Xamarin e Flutter non ha causato particolari difficoltà con la navigazione, lo stesso non si può dire di React Native. Infatti, realizzando l'applicazione con il framework di Facebook, sono stati riscontrati alcuni problemi riguardanti questo aspetto.

Gestire la navigazione con React Native è stato infatti piuttosto complicato e più volte sono sorte alcune complicazioni e difficoltà. In questi casi, si è fatto riferimento a vari forum e siti online, e si è notato che una buona parte della community di React Native riscontra spesso difficoltà legate alla navigazione, che talvolta vengono risolte aggirando il problema.

Per esempio, un problema che è stato difficile risolvere è stato il conflitto tra la libreria di navigazione [62] e quella esterna utilizzata per incorporare il video di YouTube nell'applicazione. Infatti, utilizzando un determinato metodo della libreria di navigazione, tornando alla pagina in cui veniva mostrato il video di YouTube, l'esecuzione dell'applicazione veniva interrotta. Per risolvere il problema è stato necessario sostituire la libreria per il video.

5.2.2 Inserimento video di YouTube

Un problema non di poco conto per realizzare l'applicazione è stato rappresentato dall'inserimento del video musicale di YouTube nella pagina dei dettagli della canzone. Se con Flutter non vi sono stati problemi, ed è stato sufficiente importare un plugin [64] perfettamente funzionante, con gli altri due framework sono state incontrate alcune difficoltà.

In particolare, con React Native è stato necessario testare diverse librerie esterne anche a causa dei problemi legati al conflitto con gli aspetti di navigazione, fino a quando si è riusciti a trovarne una funzionante [63].

Il framework che ha dato i problemi maggiori da questo punto di vista è stato però Xamarin, dal momento che non sono stati trovate librerie in grado di risolvere il problema in modo soddisfacente.

L'unica soluzione possibile per inserire il video all'interno della pagina con Xamarin è stata quella di utilizzare una WebView in cui incorporare il video. Tale soluzione tuttavia non è ottimale perché non consente la visione del video direttamente da MyMusic: per questioni di copyright, infatti, nell'applicazione realizzata con Xamarin il video sarà visibile solamente aprendo l'app di YouTube, come si vede nella figura 5.2.

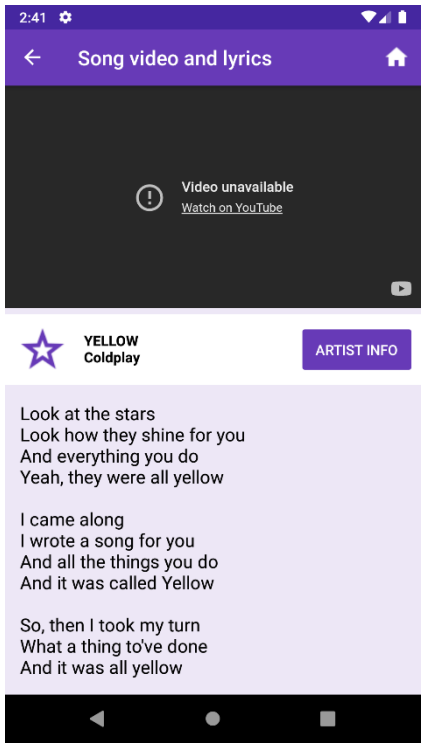

Figura 5.2: Il video di YouTube con Xamarin

5.2.3 Firestore VS Realtime Database

Un'ultima importante differenza venuta alla luce implementando l'applicazione che vale la pena di essere riportata è stata individuata nell'implementazione delle funzionalità che richiedono l'appoggio al servizio database della piattaforma Google Firebase. La versione più nuova del servizio di Firebase è Firestore, ma è ancora possibile utilizzare quella precedente, ovvero Realtime Database.

Le implementazioni di MyMusic in React Native e Flutter sfruttano i servizi di Firestore, dal momento che i framework sono stati già opportunamente aggiornati in modo da offrire servizi in grado di supportare in modo completo e perfettamente funzionante le funzionalità offerte da Firestore.

Con Xamarin, invece, si è scelto di utilizzare Realtime Database, dal momento che, pur essendo presenti librerie in grado di supportare i servizi di Firestore, essi sono ancora piuttosto nuovi e poco utilizzati, e attualmente non è presente online sufficiente documentazione o supporto per l'utilizzo, il che rende difficoltoso per utenti non esperti lo sfruttamento di tali funzionalità.

Per quel che riguarda l'implementazione delle funzionalità di MyMusic, in ogni caso, non ci sono stati problemi legati all'utilizzo di Firestore o di Realtime Database. Entrambi i servizi si sono infatti rivelati sufficienti per gli scopi dell'applicazione, e anche piuttosto simili nell'utilizzo.

Conclusione

Con il lavoro di analisi delle caratteristiche e confronto dei framework, implementazione dell'applicazione MyMusic e confronto tra le tre realizzazioni, si è riusciti a capire quali sono le principali potenzialità di Xamarin, React Native e Flutter e di quelli che invece sono i loro punti di debolezza.

Alla luce dell'esperienza accumulata nel corso del tempo e dopo aver terminato l'applicazione progettata, si è arrivati a poter affermare che, nel contesto delle funzionalità di MyMusic, il framework più adatto per lo sviluppo si è rivelato essere Flutter, che rispetto a Xamarin e React Native non ha evidenziato particolari problematiche.

Si tratta in ogni caso di una considerazione che non può essere generalizzata a qualsiasi contesto. Considerando l'applicazione MyMusic, non è stato per esempio necessario utilizzare le funzionalità hardware e software del dispositivo o altri aspetti nei quali React Native e Xamarin potrebbero essere un'alternativa migliore rispetto a Flutter.

Inoltre, la percezione che ciascuno sviluppatore ha di un framework piuttosto che di un altro potrebbe essere influenzata dalle sue conoscenze: uno sviluppatore web tenderà probabilmente a preferire React Native, mentre un programmatore Java potrebbe orientare la sua scelta verso Flutter o Xamarin. Non è quindi possibile stabilire una classifica dei framework dal migliore al peggiore, perché si tratta in ogni caso di considerazioni basate sulla propria esperienza e conoscenza.

In generale, quindi, quando si deve sviluppare un'applicazione multipiattaforma, per orientarsi nella scelta del framework più adatto devono essere tenute in considerazione per prima cosa le competenze del team di sviluppo e in secondo luogo le caratteristiche e le funzionalità dell'applicazione.

Sitografia

1. Statcounter - https://gs.statcounter.com/os-market-share/mobile/worldwide
   Data ultima consultazione: 15/07/2021

2. Forbytes - https://forbytes.com/digital-transformation/native-cross-platform-and-hybrid-app-development-what-to-choose/
   Data ultima consultazione: 08/07/2021

3. Freecodecamp - https://www.freecodecamp.org/news/a-deeply-detailed-but-never-definitive-guide-to-mobile-development-architecture-6b01ce3b1528/
   Data ultima consultazione: 12/07/2021

4. Clevertap - https://clevertap.com/blog/types-of-mobile-apps/
   Data ultima consultazione: 05/07/2021

5. Ionos - https://www.ionos.it/digitalguide/siti-web/programmazione-del-sito-web/i-vantaggi-e-gli-svantaggi-di-unapp-ibrida/
   Data ultima consultazione: 08/07/2021

6. Dzone - https://dzone.com/articles/native-vs-hybrid-vs-cross-platform-how-and-what-to
   Data ultima consultazione: 10/07/2021

7. Intrepiditservices - https://www.intrepiditservices.com/blog/hybrid-app-examples/
   Data ultima consultazione: 03/07/2021

8. Mobileappdaily - https://www.mobileappdaily.com/best-hybrid-app-frameworks
   Data ultima consultazione: 05/07/2021

9. Jscrambler - https://blog.jscrambler.com/10-frameworks-for-mobile-hybrid-apps/
   Data ultima consultazione: 05/07/2021

10. React Native - https://reactnative.dev/showcase
    Data ultima consultazione: 11/07/2021

11. Rodeoapps - https://www.rodeoapps.com/blog/cross-platform-mobile-app-development-frameworks
    Data ultima consultazione: 14/07/2021

12. Appinventiv - https://appinventiv.com/blog/cross-platform-app-frameworks/
    Data ultima consultazione: 14/07/2021

13. Fortunesoftit - https://www.fortunesoftit.com/top-5-cross-platform-app-development-frameworks-in-2021/
    Data ultima consultazione: 14/07/2021

14. Flutter - https://flutter.dev/showcase
    Data ultima consultazione: 14/07/2021

15. Microsoft - https://dotnet.microsoft.com/apps/xamarin
Data ultima consultazione: 07/07/2021

16. Microsoft - https://docs.microsoft.com/it-it/xamarin/get-started/what-is-xamarin
Data ultima consultazione: 10/07/2021

17. Microsoft - https://docs.microsoft.com/it-it/xamarin/get-started/what-is-xamarin-forms
Data ultima consultazione: 10/07/2021

18. Html.it - https://www.html.it/pag/60443/introduzione-a-xamarin-framework/
Data ultima consultazione: 10/07/2021

19. Slideshare - https://www.slideshare.net/guidomagrin/introduction-to-xamarin-48120074
Data ultima consultazione: 13/07/2021

20. Microsoft - https://docs.microsoft.com/it-it/dotnet/csharp/tour-of-csharp/
Data ultima consultazione: 12/07/2021

21. Giuneco - https://www.giuneco.tech/xamarin-binding-libraries/
Data ultima consultazione: 14/07/2021

22. React Native - https://reactnative.dev/
Data ultima consultazione: 08/07/2021

23. React Native - https://reactnative.dev/docs/getting-started
Data ultima consultazione: 08/07/2021

24. Archibuzz - https://www.archibuzz.com/blog/sviluppo-app-mobile-con-react-native-cosa-sapere
Data ultima consultazione: 12/07/2021

25. Techcrunch - https://techcrunch.com/2012/09/11/mark-zuckerberg-our-biggest-mistake-with-mobile-was-betting-too-much-on-html5/
Data ultima consultazione: 15/07/2021

26. ReactJS - https://it.reactjs.org/
Data ultima consultazione: 10/07/2021

27. Xpeppers - https://www.xpeppers.com/blog/2016/06/30/usare-react-native-nostra-ana-lisi/
Data ultima consultazione: 11/07/2021

28. Codeburst - https://codeburst.io/props-and-state-in-react-native-explained-in-simple-en-glish-8ea73b1d224e
Data ultima consultazione: 13/07/2021

29. Opencodez - https://www.opencodez.com/react-native/react-native-props-and-state.htm
Data ultima consultazione: 13/07/2021

30. Flutter - https://flutter.dev/
Data ultima consultazione: 09/07/2021

31. Flutter - https://flutter.dev/docs
Data ultima consultazione: 09/07/2021

32. Html.it - https://www.html.it/pag/367891/flutter-nuovo-framework-per-lo-sviluppo-cross-platform/
Data ultima consultazione: 10/07/2021

33. Html.it - https://www.html.it/pag/377313/il-framework-dettagli-tecnici/
Data ultima consultazione: 10/07/2021

34. Dreamsquadgroup - https://www.dreamsquadgroup.com/whats-flutter-brief-introduction/
Data ultima consultazione: 13/07/2021

35. Italiancoders - https://italiancoders.it/flutter-tutorial-introduzione/
Data ultima consultazione: 11/07/2021

36. Flutter - https://flutter.dev/docs/resources/architectural-overview
Data ultima consultazione: 09/07/2021

37. Buildflutter - https://buildflutter.com/how-flutter-works/
Data ultima consultazione: 10/07/2021

38. Devinterface - https://www.devinterface.com/it/blog/5-buoni-motivi-per-usare-flutter-nello-sviluppo-mobile
Data ultima consultazione: 14/07/2021

39. Proandroiddev - https://proandroiddev.com/flutters-compilation-patterns-24e139d14177
Data ultima consultazione: 15/07/2021

40. Dev.to - https://dev.to/jay_tillu/flutter-compilation-process-41k0
Data ultima consultazione: 15/07/2021

41. Elevatex - https://elevatex.de/how-flutter-works-under-the-hood-and-why-it-is-game-changing/
Data ultima consultazione: 11/07/2021

42. Medium - https://medium.com/xorum-io/cross-platform-mobile-apps-development-in-2021-xamarin-vs-react-native-vs-flutter-vs-kotlin-ca8ea1f5a3e0
Data ultima consultazione: 12/07/2021

43. Valuecoders - https://www.valuecoders.com/blog/technology-and-apps/flutter-vs-xamarin-vs-react-native-which-cross-platform-mobile-app-development-framework-to-choose/
Data ultima consultazione: 12/07/2021

44. Dzone - https://dzone.com/articles/flutter-vs-react-native-vs-xamarin
Data ultima consultazione: 12/07/2021

45. Italiancoders - https://italiancoders.it/flutter-vs-xamarin-confronto-tra-i-due-framework-principali-per-lo-sviluppo-mobile-cross-platform/
Data ultima consultazione: 13/07/2021

46. Dev.to - https://dev.to/logicraystech/best-cross-platform-app-development-tool-to-choose-in-2021-flutter-vs-react-native-vs-xamarin-dkg
Data ultima consultazione: 13/07/2021

47. Ionic - https://ionicframework.com/
Data ultima consultazione: 15/07/2021

48. Apache Cordova - https://cordova.apache.org/
Data ultima consultazione: 15/07/2021

49. Framework7 - https://framework7.io/
Data ultima consultazione: 15/07/2021

50. KendoUI - https://www.telerik.com/kendo-ui
Data ultima consultazione: 15/07/2021

51. OnsenUI - https://onsen.io/
Data ultima consultazione: 15/07/2021

52. Quasar - https://quasar.dev/
Data ultima consultazione: 15/07/2021

53. Aurelia - https://aurelia.io/
Data ultima consultazione: 15/07/2021

54. ExtJS - https://www.sencha.com/products/extjs/
Data ultima consultazione: 15/07/2021

55. CapacitorJS - https://capacitorjs.com/
Data ultima consultazione: 15/07/2021

56. Appcelerator - https://www.appcelerator.com/
Data ultima consultazione: 15/07/2021

57. Nativescript - https://nativescript.org/
Data ultima consultazione: 15/07/2021

58. Weex - http://emas.weex.io/
Data ultima consultazione: 15/07/2021

59. Svelte Native - https://svelte-native.technology/
Data ultima consultazione: 15/07/2021

60. Rubymotion - http://www.rubymotion.com/

Data ultima consultazione: 15/07/2021

61. Solar2d - https://solar2d.com/
Data ultima consultazione: 15/07/2021

62. React Navigation - https://reactnavigation.org/
Data ultima consultazione: 17/07/2021

63. Npmjs - https://www.npmjs.com/package/react-native-youtube-iframe/
Data ultima consultazione: 17/07/2021

64. Pub.dev - https://pub.dev/packages/youtube_player_flutter
Data ultima consultazione: 17/07/2021

65. API Genius – https://docs.genius.com/
Data ultima consultazione: 17/07/2021

66. lyrics.ovh - https://lyricsovh.docs.apiary.io/
Data ultima consultazione: 17/07/2021

67. Firebase – https://firebase.google.com/
Data ultima consultazione: 17/07/2021