Introduzione

La sfida tra approcci diversi alla missione ultima di realizzare un'app mobile rimane qualcosa di estremamente complesso da analizzare. Gli approcci sono sempre stati caratterizzati da due tipi differenti: ibrido e nativo. Almeno negli ultimi 10 anni, e ancora oggi, il mercato, le aziende, gli sviluppatori e tutte le parti in causa che ruotano attorno allo sviluppo mobile valutano quale strada sia meglio scegliere, e questa decisione è sempre dipesa da tanti fattori; tra questi, il principale rimane, come preponderante, quello economico.
Un approccio nativo, almeno in termini aziendali, prevede due filoni di sviluppo differenti. Sviluppare un'app mobile nativamente significa sviluppare un'applicazione in Objective-C o Swift per iOS e Java o Kotlin per Android.
Per quanto sia possibile trovare sviluppatori che governano entrambe le skill necessarie allo sviluppo per i due principali sistemi operativi mobili, rimane certamente una sfida piuttosto ardua che non tutte le aziende scelgono di intraprendere. È indubbio che le applicazioni native siano più efficienti, in primis con performance superiori, di quelle ibride; ma, al contempo, è altrettanto incontestabile che quelle ibride risultino più sostenibili, sia in fase di sviluppo che di manutenzione.
Queste considerazioni sono, con tutta probabilità, avvalorate da tutte le alternative nate nell'ultimo decennio; e tra tutte siamo qua a parlare di quella che a oggi è tra le più promettenti: Flutter (`https://flutter.dev`)

Figura I.1 Logo Flutter.

NOTA

Analizzare e confrontare le performance tra nativo e ibrido significa analizzare anche le performance native divise per linguaggi differenti. Uso della CPU, GPU, Memory, FPS e consumo della batteria sono alla base di un benchmark completo. Alcuni benchmark vedono Flutter più performante di altre alternative ibride ed estremamente competitivo con applicazioni native.

Nativo e ibrido

Nel caso in cui alcuni di noi si stiano chiedendo quale effettivamente sia la differenza tra app native e app ibride, apriamo una brevissima parentesi, in attesa di un approfondimento nei capitoli successivi. Le app native si interfacciano direttamente con l'ambiente che le ospita (sistema operativo) e vengono sviluppate con un software (IDE) specifico. Le app ibride, invece, in genere necessitano di uno strato intermedio, che introduce un calo delle performance e di conseguenza dei compromessi che introducono problematiche legate alla user experience e in contesti specifici anche delle vere e proprie limitazioni. Possiamo semplicemente immaginare un concetto molto banale. L'app nativa consente allo sviluppatore di "dialogare" con il sistema operativo mobile attraverso un sdk, che consente in tutto e per tutto la gestione del dispositivo. L'approccio ibrido impone l'utilizzo di un framework di terze parti, che consentirà di intermediare le funzionalità offerte dal sistema operativo, o meglio quelle che il framework implementa.
Negli anni ogni framework nato per lo sviluppo di applicazioni mobili ibride si è prefissato lo scopo di sviluppare un software che consentisse, con un unico ambiente di sviluppo, di creare un'applicazione mobile per più sistemi operativi. La sfida si è sempre caratterizzata sull'implementazione, che risulta essere una vera e propria firma del framework.

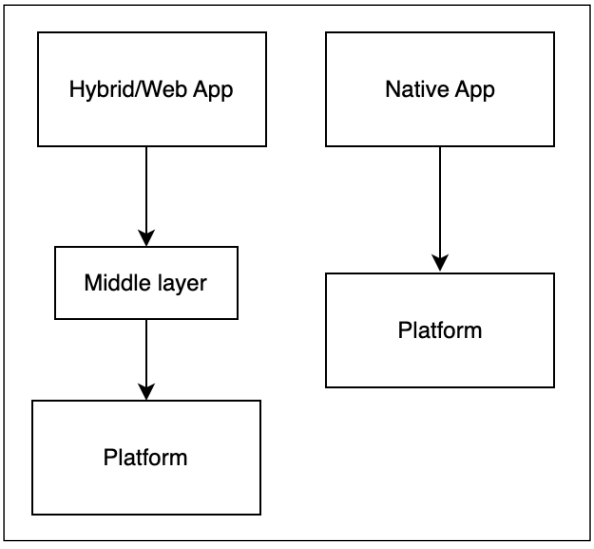

Figura I.2 Differenza sommaria tra app native ed ibride.

I framework ibridi sostanzialmente si sono divisi negli anni tra approcci differenti. Per esempio, uno dei primi framework ibridi, chiamato Cordova, utilizzava un layer intermedio per renderizzare un applicativo web all'interno di una webview e utilizzava Cordova Plugins per dialogare con il sistema operativo.
React Native, vera alternativa a Flutter, utilizza un layer intermedio (Fabric e TurboModules nella nuova architettura) per renderizzare l'app e dialogare con il sistema operativo.

NOTA

Per webview intendiamo un componente nativo che consente di eseguire un browser in-app. Di fatto, è possibile visualizzare applicativi web all'interno della nostra app mobile.

L'architettura di Flutter, a differenza delle altre alternative ibride citate, non implementa nessun layer intermedio; ciò consente un incremento delle prestazioni. Il framework Flutter implementa direttamente le interfacce di UI, chiamate widget e, oltre a questo, il motore di rendering utilizzato da Flutter disegna pixel per pixel la user interface sullo schermo del device. React Native deve interfacciarsi con i componenti nativi per disegnare la UI; altre alternative ancora utilizzano il browser. Insomma, questo step ulteriore è in buona parte una caratteristica distintiva tra Flutter e le alternative.
In Figura I.2 abbiamo riassunto con Platform il layer che identifica il sistema operativo. La differenza tra le architetture, rimanendo ad altissimo livello, è che Platform può essere immaginato come composto a sua volta da due differenti blocchi: la parte relativa al rendering e quello che offre l'integrazione con i servizi e corrispettivo hardware. Mentre React Native, Xamarin o NativeScript dialogano con il sistema operativo per il rendering dei componenti nativi, Flutter salta a piè pari questo step in quanto utilizza i propri widget e si preoccupa lui stesso della fase di rendering.

NOTA

React Native, Xamarin, NativeScript, Ionic sono a oggi i più comuni framework per applicazioni mobili con architetture differenti e differenti logiche.

Basti pensare che già l'utilizzo di una UI nativa, come offerto da alcuni competitor, è stato un grande passo avanti rispetto agli esordi delle app ibride dove la UI era rappresentata all'interno di un browser. Credo che sarà il tempo a decidere se questa scelta distintiva presa dal team di Flutter sarà stata vincente; io credo di sì e a oggi i numeri legati all'adozione del framework lo confermano.
Le diverse tecnologie obbligano lo sviluppatore a doversi adattare ai differenti framework. Lo sviluppatore segue rigidamente l'ambiente di sviluppo su cui si basa il framework, utilizzando tutti gli strumenti offerti, e tecnologie tanto differenti richiedono conoscenze specifiche che difficilmente porteranno lo sviluppatore a padroneggiarle tutte.
Oltre all'uso del framework in sé, Flutter pone lo sviluppatore dinanzi alla sfida di un nuovo linguaggio di sviluppo. Molti framework negli anni si sono basati sul linguaggio "web": JavaScript, con la possibilità di utilizzare il superset TypeScript. Flutter, invece, si basa su un linguaggio sviluppato di recente: Dart.

NOTA

Conosciuto inizialmente con il nome di Dash, Dart è un linguaggio di programmazione sviluppato da Google nel 2011 con l'obiettivo di sostituire JavaScript.

L'applicativo ibrido, qualunque esso sia, se sviluppato con destrezza potrebbe mascherare all'utente finale che l'applicativo in uso non è nativo. Per quanto la user experience possa avvicinarci il più possibile a quella nativa, il problema principale si annida nelle performance. La nascita di framework per applicazioni mobili ibridi si è sempre susseguito cercando di limare le problematiche legate principalmente alle performance. La tecnologia che è stata leader del settore per anni, Apache Cordova, sta venendo sempre di più soppiantata

dalle "nuove alternative" proprio perché l'utilizzo di una webview con un rendering basato su HTML/CSS non è una soluzione che, con la nascita di applicazioni sempre più complesse, riesce a soddisfare la user experience che oggi richiede l'utente finale.

NOTA

Apache Cordova è un framework rilasciato open source da Adobe che a sua volta si basava su PhoneGap creato nel 2009.

Anche spostandosi su soluzioni che non si basano su un rendering "web", un layer intermedio che si preoccupa di dialogare con un componente nativo, indipendentemente dal framework utilizzato, potrebbe comunque rendere l'app macchinosa, oltre che lenta. Immaginiamo app complesse: quanto dispendio di memoria è richiesto per poter renderizzare tutti gli elementi grafici che compongono il layout? Flutter si pone esattamente in questo segmento di mercato, offrendo una soluzione che garantisce performance elevate e una curva di apprendimento non troppo ampia.
Per quanto l'uso di Flutter possa essere comparato con le sue alternative, come già anticipato, rispetto all'utilizzo del nativo, chiaramente l'aspetto vincente si basa sul Time To Market, che altro non è che il tempo impiegato dall'inizio dello sviluppo di un prodotto alla sua "commercializzazione". Statistiche di cui non sono autore, ma che reputo verosimili, parlano di tempi di sviluppo dimezzati del 40/50% rispetto allo sviluppo nativo sulle due piattaforme. Inoltre, oltre ai tempi di sviluppo, si aggiungono quelli di maintenance e testing che a loro volta vengono dimezzati. Insomma, per essere schietto e sincero, è più difficile trovare dei motivi realmente validi per approcciare il nativo, che viceversa; e sono fermamente convinto che Flutter potrebbe, nel futuro, minare realmente quello che a oggi rimane territorio prettamente nativo: games, big data, UI altamente customizzate, alte prestazioni o app che necessitano di aspetti specifici legati al sistema operativo.
La sfida resta aperta e c'è chi ha già scommesso che in futuro Flutter soppianterà di fatto lo sviluppo nativo, ma non ditelo ad Apple.

Punti di forza

Flutter, rispetto a tutte le soluzioni oggi presenti nel mercato, riesce a essere particolarmente performante. Rispetto a competitor che utilizzano uno strato intermedio di comunicazione con il sistema operativo, Flutter compila direttamente l'applicativo in linguaggio macchina e questo dialoga direttamente con il sistema operativo e i vari componenti. Un approccio di questo tipo rende efficiente l'applicativo e le sue performance sono molto vicine a quelle che avremmo con uno sviluppo nativo. Flutter, a oggi, è sicuramente l'alternativa più performante.

Benchmark

Riporto i dati di un articolo molto ben fatto di una società ucraina (inVeritaSoft), che ha realizzato diversi benchmark per sistema operativo, linguaggio e casi d'uso differenti.

NOTA

Di seguito, gli URL dei benchmark: `https://inveritasoft.com/blog/flutter-vs-react-native-vs-native-deep-performance-comparison`, `https://inveritasoft.com/blog/flutter-vs-native-vs-react-native-examining-performance`

Dalla comparazione si evince sostanzialmente quello che ci si aspettava. La scelta nativa rimane la migliore in termini di prestazioni in caso di animazioni pesanti, ma Flutter dimostra di essere molto più performante delle alternative, proprio per quello che ci siamo detti appena sopra.
Consultando altri benchmark, per esempio `https://blogs.perficient.com/2020/11/02/android-app-native-vs-flutter-vs-react-native/` o `https://www.orientsoftware.com/blog/flutter-vs-react-native-performance/` si raggiunge ugualmente la stessa conclusione.

Look and feel

Un altro aspetto che si differenzia dalle alternative è la gestione dei widget. In Flutter,ogni cosa è un widget. Ogni parte, ogni comportamento, che compone il layout è un widget. Tutto il codice che scriveremo vivrà all'interno di un widget. I widget, nonostante possano essere personalizzati, sono di proprietà di Flutter e questo, come scopriremo, è un bene.

NOTA

Per chi storce il naso poiché preferisce l'approccio *self made*: sì, è possibile creare i propri widget. L'uso dei widget creati dalla community è anch'essa una pratica comune, ma, a differenza di altri package manager, l'ambiente usato da Flutter ha creato dei criteri valutazione dei package che attenzionano e aiutano lo sviluppatore nell'adozione di quest'ultimi.

In genere, non dovremo preoccuparci di come questi appaiono e se funzionano correttamente su sistemi operativi mobili differenti, e vi garantisco che non è poca cosa.
Il cosiddetto *look and feel* è garantito da Google stessa. Flutter si occuperà di gestire alcuni comportamenti che sono specifici per sistema operativo, come la navigazione, in cui troviamo per esempio una differenza per l'animazione della transizione tra una view e l'altra. L'animazione in questione è effettivamente diversa tra Android e iOS, ma è solo un esempio tra i tanti. Oltre a questi comportamenti, specifici per OS, l'uso dei widget portano con sé un design predefinito; i nostri esempi si baseranno sul Material design, sia per iOS che per Android. Flutter offre comunque la possibilità di applicare stili differenti per sistema operativo; per esempio, offre una serie di widget con design Cupertino (iOS-style), che potranno essere utilizzati in ambiente Apple.
Non dover spendere una marea di energie sul *look and feel* tra i vari platform è davvero tanta roba. Per non parlare dell'hot reload, una feature fondamentale durante la fase di sviluppo, che consente di ricompilare immediatamente il codice modificato rendendo fruibile la modifica all'interno dell'app.
Un altro aspetto che caratterizza Flutter è l'utilizzo di Skia. Questo motore grafico consente di renderizzare, molto più velocemente rispetto ad altre alternative, il layout. L'esperienza di rendering, oltre a non doversi premurare di come questa avvenga sulle diverse piattaforme, renderà l'applicazione sarà incredibilmente fluida e conforme.

NOTA

Skia è una libreria grafica 2D open source. Acquisito da Google nel 2005, questo progetto è diventato il motore grafico 2D di importanti tools, come Google Chrome e Firefox.

Infine, Flutter non è solo relegato al mondo mobile, ma può essere utilizzato anche per applicativi web e desktop. Questo aspetto è una feature molto potente e non del tutto comune, quanto meno con la facilità con cui è possibile farlo in Flutter. Il framework consente con estrema facilità di compilare ed eseguire un'applicazione su platform come Windows, macOS, Linux, piuttosto che un browser, senza che ci sia la necessità di modificare il codice sorgente.
Questi aspetti peculiari, che verranno approfonditi nel dettaglio nei capitoli successivi, hanno reso Flutter un reale competitor di React Native che, fino a poco tempo fa, dominava incontrastato il mercato.

Adoption

Flutter è nato a metà del 2017 e, nonostante si basi su un linguaggio non largamente diffuso come Dart, ha fatto molti passi avanti in termini di adozione, soprattutto negli ultimi due/tre anni. Nel survey di stackoverflow del 2022 si può osservare quanto sia diffuso. Rispetto ai due anni precedenti, la curva di adozione è notevole: un aumento del 5% di progetti che usano Flutter (Figura I.3 e Figura I.4).
Google Trends può venirci in aiuto nel capire quanto effettivamente la curva di interesse su Flutter, dagli addetti ai lavori e non solo, sia aumentata nel tempo, tanto da dividersi il mercato con quella che a oggi era la reale alternativa allo sviluppo di applicazioni ibride.

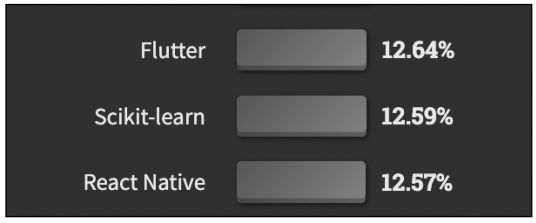

Figura I.3 Stack Overflow Developer Survey 2022.

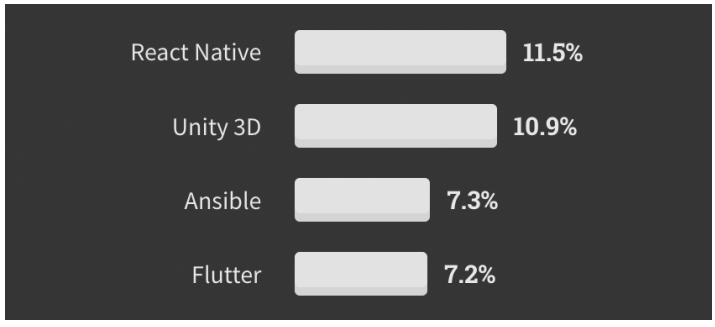

Figura I.4 Stack Overflow Developer Survey 2020.

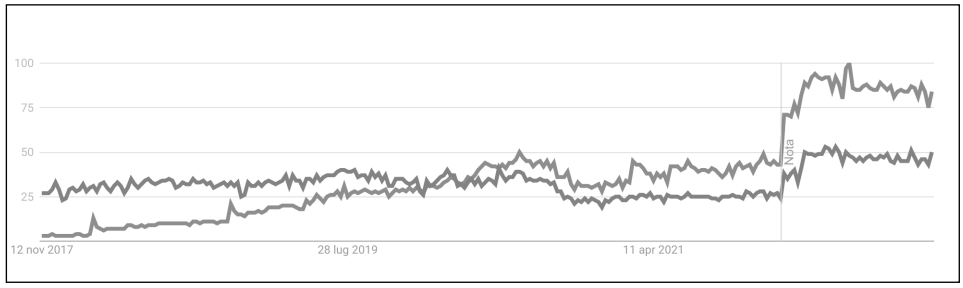

Figura I.5 Google Trends mostra quanto Flutter (blu) partito da 0 nel 2007 nel giro di pochi anni abbia invertito il trend rispetto a React Native (rosso).

A chi è rivolto Flutter

L'attuale panorama legato al mondo delle app ibride è piuttosto vasto, ma a farla da padrone è senz'altro React Native, con Ionic a seguire (anche se molto dietro). Ionic è un altro framework per app ibride che si basa su Cordova.
React Native è il framework che offre prestazioni più elevate rispetto ai suoi competitor e ha sostituito Ionic come framework di riferimento per questa categoria di progetti. Gli sviluppatori coinvolti su progetti cross-platform hanno una naturale confidenza con React Native in quanto il linguaggio di programmazione di riferimento è JavaScript e la familiarità è fondamentale soprattutto nello start di un nuovo progetto. In aggiunta, React Native si basa su React che, come sappiamo, è un noto framework per applicazioni web. Detto ciò, chi ha già familiarità con React non può che saltare a bordo di React Native per lo sviluppo di un progetto mobile.

> **NOTA**
>
> React Native è un framework Open Source sviluppato da Meta (ex Facebook) nel 2015.

React Native si basa su una logica in cui tra il sistema operativo e il framework c'è un layer intermedio (come in Figura I.1). Lo strato nativo dialoga con quello javascript tramite questo layer, che di fatto è il cuore della tecnologia che sta alla base di React Native. Come abbiamo già anticipato sommariamente, uno dei punti di forza di Flutter è la compilazione, che rende totalmente inutile l'utilizzo di strati intermedi di comunicazione. Google, con Flutter, si è posta nel bel mezzo di questa fetta di mercato creando una nuova tecnologia, sfruttando così a pieno la possibilità di realizzare app cross-platform con un paradigma diverso rispetto a quanto già offerto.
L'obiettivo primario di questa opera è senz'altro quello di rivolgersi alla più ampia platea possibile. Per quanto molti di noi rimangono degli intramontabili utilizzatori del desktop, è insindacabile che un applicativo mobile è più importante di qualsiasi altra cosa. D'altronde, l'approccio Mobile first è quello che ha caratterizzato gli ultimi anni. Prima il device mobile nella progettazione di un applicativo web o mobile app, poi il desktop. L'approccio Mobile first non ha dettato nessuna regola, ha semplicemente seguito il suggerimento del mercato. Oggi, gli utenti che navigano dal dispositivo mobile sono il 20% in più di quelli che usano il desktop; questo è quanto (Figura I.6).

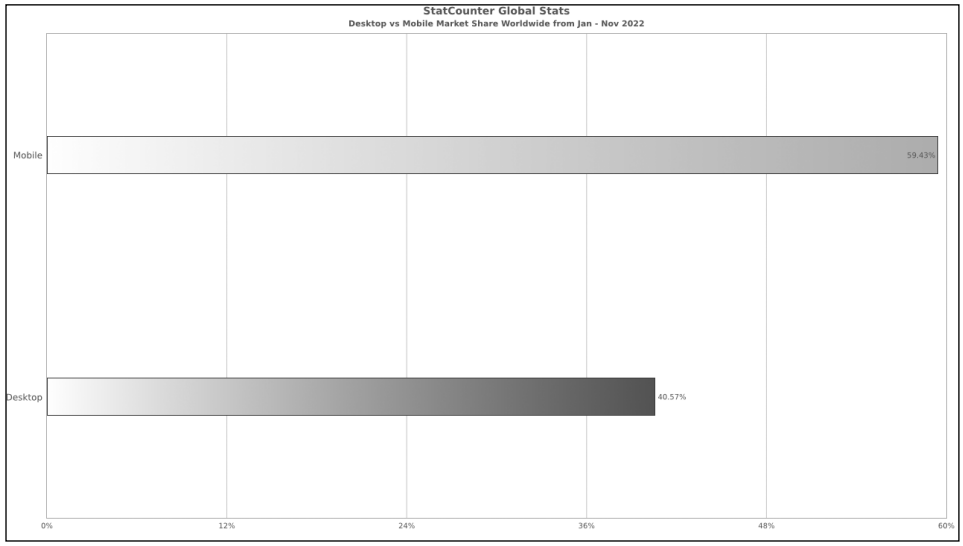

Figura I.6 Statiche uso dispositivi mobili by Statcounter.com.

Un'amara conclusione, dettata semplicemente dall'anzianità di chi scrive, è che sono più le applicazioni mobile che non hanno un corrispettivo web che viceversa. Flutter, tutto sommato, viene in aiuto con la sua feature di compilazione cross-platform anche per i futuri nostalgici o per chi ha necessità legate esplicitamente alla navigazione di un sito internet.

> **NOTA**
>
> Nel caso dello sviluppo di un sito web piuttosto complesso non è detto che la tecnologia ibrida rimanga comunque valida. Il framework Angular, a mio parere, rimane la scelta migliore per applicativi web complessi che trovano limitativo l'uso esclusivo del mobile.

Le esigenze di business hanno tracciato la direzione, ma già da parecchi anni si predilige un'attenzione maggiore al supporto mobile. Le aziende e gli sviluppatori mantengono sempre alta la soglia di interesse per tutte le innovazioni tecnologiche che portano valore aggiunto. In questi anni, però, molte aziende hanno fatto a proprio spese l'esperienza di aver scelto la strada dell'approccio ibrido senza analizzare nel dettaglio quello che la tecnologia offriva. Applicazioni complesse e ibrido è un connubio che può rivelarsi pericoloso. Aziende importanti come, per esempio, AirBnb, hanno fatto la scelta di tornare al nativo dopo una breve e poco redditizia esperienza con React Native.
Altre aziende, invece, mantenendo uno standard di sviluppo qualitativamente elevato, sono riuscite a implementare layout, logiche di business e user experience molto complesse in React Native con grande successo; un esempio è Uber eats.
Dal mio punto di vista, JavaScript e il compromesso nell'utilizzo di un layer intermedio limitano di molto quello che Google con Flutter ha provato ad arginare: nuovo linguaggio e nessun layer intermedio.
Performance, *Code Development Time* e *time to market speed* sono argomenti che mettono d'accordo sviluppatori e aziende. Una curva di apprendimento bassa garantisce un approc-

cio veloce; soprattutto, un framework come Flutter, con le sue features legate all'utilizzo dei widget, non necessita di skill elevate e profondamente mature: mi riferisco all'uso di JavaScript in progetti di grosse dimensioni. Quando si parla di progetti importanti, in cui gli sviluppatori riescono a portare a casa un buon risultato senza aver perso anni di vita lungo il percorso, significa che abbiamo trovato la giusta confidenza con la tecnologia.
La sfida che mi pongo con questo volume è proprio questa. Neofita, sviluppatore nativo, sviluppatore ibrido, scopriamo insieme il valore aggiunto di Flutter e comprendiamo insieme i suoi punti di forza e le sue debolezze, fino a capire perché Google si sia mossa in questa direzione.
Il livello di dettaglio di questo volume sarà piuttosto alto cercando di non dare nulla per scontato, provando a garantire una facile lettura e comprensione, anche per i non avvezzi allo sviluppo. In poche parole sarà una sfida piuttosto ardua.
La tecnologia in questione è recente, quindi sarà qualcosa di nuovo sia per uno sviluppatore che per un novellino. Muoveremo i primi passi verso Flutter, approfondendo i lati più tecnici e interni alla tecnologia, e studieremo il linguaggio di sviluppo Dart. Durante il percorso, insieme, svilupperemo la nostra prima vera app.

I revisori

Ralf Mureddu è Software Engineer specializzato in sicurezza applicativa con esperienza più che decennale in ambito finanziario.
Carlo Zappatini è Software Engineer specializzato in applicazioni mobile e front-end con un passato da programmatore embedded in ambito automotive.

Il codice

Tutto il codice sorgente presente in questo libro è disponibile all'indirizzo `https://bit.ly/apo-saf` e su GitHub all'indirizzo `https://github.com/vgflutter/flutter_apogeo`.

Ringraziamenti

Il ringraziamento principale va ai lettori, che spero con tutto il cuore siano soddisfatti. Non è la mia prima esperienza in questo campo, tutt'altro, e ogni lavoro rende il prossimo sempre più sfidante. Alzare l'asticella nella speranza di far sempre meglio è l'obiettivo che mi sono prefissato. Nonostante l'ultimo anno non sia stato particolarmente felice, ho portato avanti il lavoro con il massimo impegno e non ho rimpianti: ho fatto del mio meglio. Quindi, grazie a voi per aver dedicato tempo e attenzione al mio lavoro.
Un altro grazie ai miei fedelissimi: a mia moglie per la pazienza e alla coppia Ralf Mureddu e Carlo Zappatini per il lavoro di revisione che hanno condotto cercando di complicarmi la vita il più possibile. Grazie davvero.
Infine, un saluto a tutti i colleghi di Cornèr Banca SA. Nonostante la quotidianità lavorativa sia in antitesi con il lavoro da autore, un ambiente di lavoro stimolante come quello che mi trovo a vivere giornalmente aiuta non poco a mantenere alto il morale e la voglia di fare.

Detto ciò, credo fermamente che Flutter diventerà il riferimento principale nel mondo del cross-platform development, e mi auguro che riguardo il tema Flutter questo sia solo un arrivederci.
Contattatemi pure via Linkedin https://www.linkedin.com/in/vincenzo-giacchina-1550538b o via mail (giacchina.vincenzo@gmail.com): sarò felice di ricevere feedback, domande e, nel mio piccolo, proverò a darvi supporto.
Grazie ancora e a presto.