Relazione per il progetto di Mobile Programming e Multimedia

Redazione | Nicola Agostini

Descrizione
Questo documento descrive il progetto svolto per il corso di Mobile Programming e Multimedia della laurea magistrale in informatica

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

1.1 Descrizione del progetto

Il progetto che ho sviluppato per il corso di mobile programming e multimedia ha riguardato lo sviluppo di due diverse applicazione cruscotto utilizzando due diversi framework cross-platform. Le due applicazioni realizzate hanno lo scopo di mostrare all'utente, attraverso un'interfaccia semplice ed intuitiva, i valori dei principali sensori del proprio smartphone. Sono stati utilizzati due framework a me sconosciuti con lo scopo di imparare nuovi linguaggi e framework e poi confrontarli con quelli conosciuti.

1.2 Framework utilizzati

Sono stati utilizzati **Flutter** e **Xamarin.Forms**. Flutter è stato scelto perché è un framework rilasciato recentemente da google ed avevo intenzione di imparare le basi dato che il suo utilizzo sta rapidamente crescendo. Ho scelto Xamarin.Forms poiché è uno dei framework per sviluppo cross-platform più diffusi al momento e particolarmente adatto alla creazione di app cruscotto, inoltre non lo conoscevo e ho voluto colmare questa importante lacuna.

2 Flutter

Flutter è un framewrok cross-platform per la creazione di applicazioni sia per IOS che per Android. E' stato rilasciato da Google in *Alpha* nel maggio del 2017, l'ultima versione ora è la *1.0 stabile* rilasciata il 4 dicembre 2018. Vediamo l'architettura generale di Flutter:

- **Dart:** le applicazioni sono scritte con il linguaggio Dart, anch'esso sviluppato da Google, che viene eseguito, grazie alla *Dart virtual machine*, just-in-time; inoltre fornisce la possibilità di utilizzare l'hot reloading cioè una volta apportate le modifiche al codice è sufficiente salvarle per vederle subito apportate al proprio simulatore dove si sta testando il codice. La funzionalità dell'hot reloading è stata molto utile nello sviluppo del progetto in quanto si perde molto meno tempo rispetto a dover ricompilare tutto quando si vuole vedere delle modifiche apportate al codice soprattutto riguardanti il layout;
- **Flutter engine:** scritto in *C++* offre supporto alla renderizzazione grafica di base tramite la libreria grafica *Skia*; inoltre si interfaccia con l'SDK specifico sia di Andorid che di IOS;
- **Foundation library:** libreria scritta in Dart che mette a disposizione varie funzioni e classi per creare e manipolare gli oggetti grafici nell'interfaccia dell'applicazione;
- **Widget specifici:** vi sono due categorie distinte di widget per i due sistemi operativi: i *Material Design* widgets per Android e i *Cupertino* widgets per IOS. Sono molto utili in quanto si può costruire velocemente un'interfaccia completa e moderna per avere subito un'idea di come appare l'applicazione.

In Flutter "tutto è un widget" e un widget può avere uno stato che ne determina contenuto o forma. Per esempio l'applicazione è un widget che a sua volta contiene un'appbar (che è un widget) e una parte centrale (che è anch'essa un widget) che sua volta contiene altri widget (bottoni, righe, ecc..). Vi sono due tipi principali di widget quelli privi di stato, che sono immutabili, e quelli dotati di stato, che rispondono alle varie interazioni, modifiche.

Inoltre, Flutter, viene eseguito diversamente quando è in fase di debug rispetto a quando è stato buildato e rilasciato. In fase di debug vengono lanciati nel device il *Dart runtime*, il compilatore e interprete *Just-in-Time* e lo strumento per il debugging; tutto ciò appesantisce molto l'applicazione ma è essenziale in fase di debug e per permettere l'hot reloading. Nella *release mode* invece rimane solamente il *Dart runtime* che include il garbage collector per rimuovere gli oggetti diventati inaccessibili ed inutilizzabili. Quindi secondo la classificazione dei framework, Flutter in *debug mode* corrisponde ad un approccio interpretato, ma quando viene fatta la build allora corrisponde ad un'applicazione *cross-compiled*.

2.1 Pregi

Hot Reload Eseguendo l'applicazione in sviluppo su un emulatore è possibile testare i cambiamenti al codice salvandoli e vedendo, nel giro di pochi secondi, il cambiamento già effettuato all'applicazione in prova sull'emulatore.

Prototipazione Essendo molto veloce lo sviluppo e consentendo di utilizzare il material design per la grafica e un layout facile ed intuitivo, Flutter si presta molto bene allo sviluppo di prototipi e MVP (minimum viable product).

Widget Il fatto che tutte le componenti siano dei widget permette il loro riuso, personalizzazione e facilità di creazione delle applicazioni. Inoltre è possibile scrivere codice nativo per accedere alle funzionalità specifiche del dispositivo.

Supporto Al momento vi sono moltissimi sviluppatori, non solo di Google, che stanno contribuendo alla crescita del framework. Quindi Flutter è in rapida crescita e ha una comunità molto attiva che lo supporta. Inoltre è molto facile da configurare per via della documentazione dettagliata e delle guide presenti sul web. Durante lo sviluppo dell'applicazione ho trovato molto supporto su vari siti come *Stack Overflow* in cui ho potuto risolvere i problemi comuni tra gli sviluppatori neofiti come me.

2.2 Difetti

Librerie disponibili Personalmente ho trovato poche librerie disponibili per l'accesso ai sensori del dispositivo; in generale le librerie per Flutter, al momento, sono poche e sono disponibili al sito `https://pub.dartlang.org/`. Infatti siccome l'applicazione utilizza vari sensori, ho trovato solamente librerie in grado di leggere i valori dell'accelerometro, giroscopio e GPS. Comunque rimane il fatto che è sempre possibile scrivere codice in nativo ed integrarlo nel proprio progetto flutter ma così si perde il vantaggio di utilizzare un framework cross-platform per sviluppare sia su IOS che su Andorid.

Dart Nonostante il linguaggio Dart comporti vari pregi come la possibilità della compilazione Just-in-Time, è abbastanza ostico e complesso da imparare all'inizio per esempio presenta alcuni costrutti abbastanza complicati come gli *Stream* che per impararli richiedono abbastanza tempo.

3 Xamarin.Forms

Xamarin.Forms nasce dall'unione di due progetti: Xamarin.IOS e Xamarin.Andorid. E' un framework pensato principalmente per le interfacce dotandole di *look and feel* nativo, l'interfaccia può essere realizzata utilizzando C# oppure XAML che è un linguaggio XML in cui attraverso l'uso di vari tag si crea la struttura dell'interfaccia. Xamarin.Forms offre due diversi approcci per le app con IOS e con Android; per IOS utilizza Mono per compilare il codice scritto in C# in linguaggio assembly ARM che viene utilizzato dal nostro dispositivo IOS. Le librerie native assieme al codice compilato sono impachettate assieme a formare il file IPA che può essere rilasciato sull'App Store. Per il fatto che Apple non permette la compilazione JIT, quindi richiede che Xamarin faccia uso della compilation AOT. Per quanto riguarda Xamarin.Android, le applicazioni vengono eseguite all'interno dell'ambiente di esecuzione di Mono e non richiede compilazione AOT. Viene eseguita assieme alla macchina virtuale runtime di Andorid (ART), entrambi gli ambienti sono eseguiti sul kernel Linux ed espongono diverse API utilizzabili dagli sviluppatori.

3.1 Pregi

XAML Molto veloce nella costruzione di applicazioni base, viene rimossa la complessità presente nella codifica in nativo. Lo sviluppo dell'applicazione procede molto velocemente e XAML non presenta alcuna difficoltà nell'apprendimento in quanto è praticamente identico ad un qualsiasi XML.

MVVM Applicando il design architetturale Model-View-ViewModel diventa molto facile e logico controllare i cambiamenti della vista in relazione ai cambiamenti dello stato, il tutto comporta facilità e organizzazione nello sviluppo rendendo il codice più strutturato e facilmente comprensibile.

Community La community di Xamarin.Forms è molto vasta e attiva, molto più di quella di flutter. Nello sviluppo dell'applicazione ho trovato varie guide, tutorial ed esempi di applicazioni per imparare.

Librerie Sono presenti moltissime librerie per Xamarin.Forms tra cui *Xamarin.Essential* per l'accesso ai sensori dello smartphone che ho utilizzato per creare l'applicazione.

3.2 Difetti

Funzionalità limitate Se proprio bisogna trovare un difetto a Xamarin.Forms è il fatto che se voglio funzionalità più specifiche e mirate, devo integrare codice nativo. Comunque ciò avviene praticamente in tutti i framework cross-platform e quindi non è un svantaggio unicamente riservato a Xamarin.Forms.

3.3 Riferimenti

- **Flutter:**
  https://en.wikipedia.org/wiki/Flutter_(software)
  https://italiancoders.it/flutter-sviluppo-mobile-cross-platform-firmato-google/
  https://medium.com/flutter-io/flutter-dont-fear-the-garbage-collector-d69b3ff1ca30
  https://hackernoon.com/flutter-pros-and-cons-for-seamless-cross-platform-development-c81bde5a4083
- **Xamarin.Forms:**
  https://medium.com/@samstone/pros-and-cons-of-xamarin-native-and-xamarin-forms-development-from-a-na
  https://tim.klingeleers.be/2017/05/05/look-under-hood-xamarin-forms/

4 DashDroid Flutter

Questa sezione descrive l'applicazione DashDroid e le principali scelte di progettazione, per il codice si rimanda al repository Github `https://github.com/P4rTY6/DashdroidFlutter.git` .

4.1 Home page IOS

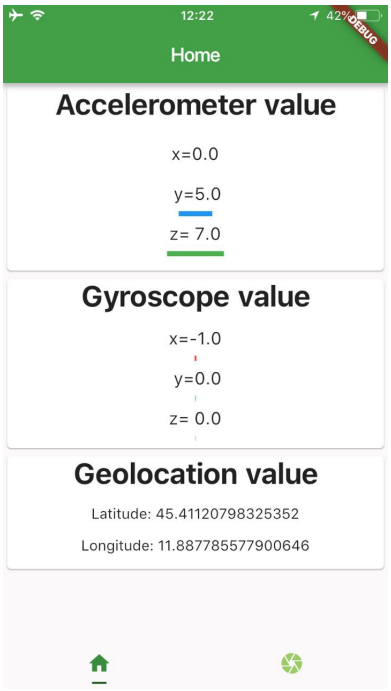

Figura 1: Home page IOS

Nella figura 1 è presente la home page su IOS in cui in alto vi è la *TopBar* in cui è presente il nome dalla pagina corrente. Al centro troviamo 3 card una per il valore dell'accelerometro una per quella del giroscopio e l'ultima contenente la posizione GPS. In basso è presente una *Bottom Navigation* con 2 pulsanti, uno per la pagina della home e l'altro per navigare alla pagina della fotocamera. Secondo le indicazione di layout per IOS i due pulsanti per navigare nell'applicazione sono in basso, nella *confort zone* appunto poichè saranno due pulsanti premuti spesso e non portano ad azioni pericolose tali per cui dovrebbero venire posizionati fuori dalla confort zone.

4.2 Home page Android

La figura 2 mostra la home page su Android in cui le card con i valori dei sensori sono esattamente identiche, cambia solamente il tipo di navigazione poichè su android abbiamo in basso la barra del sistema operativo è sconsigliato porre bottoni di navigazione sopra la barra del sistema operativo perchè si rischierebbe di premere il pulsante sbagliato avendo più bottoni molto vicini tra loro. La strategia di navigazione adottata è la *Drawer Navigation* che appunto consiste nell'aprire il menu di navigazione premendo il bottone con le 3 linee orizzontali in alto a sinistra. Questo bottone, in dispositivi dal display abbastanza grande, è fuori dalla confort zone, ma ciò non reca alcun problema per il fatto che comunque sono abilitate le gesture; ovvero effettuando uno *swipe* verso destra si aprirà sulla sinistra il menu in figura 3.

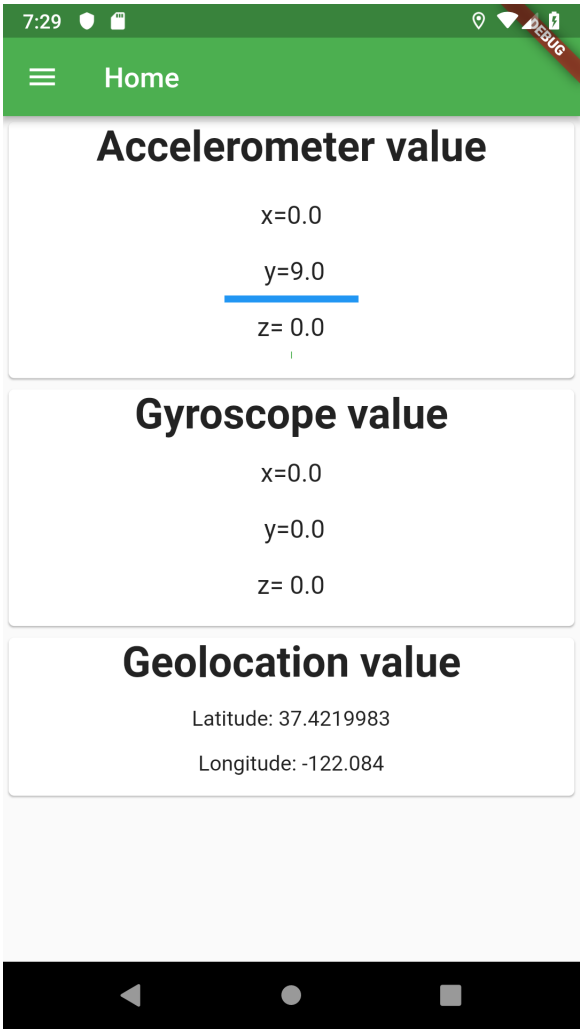

Figura 2: Home page Android

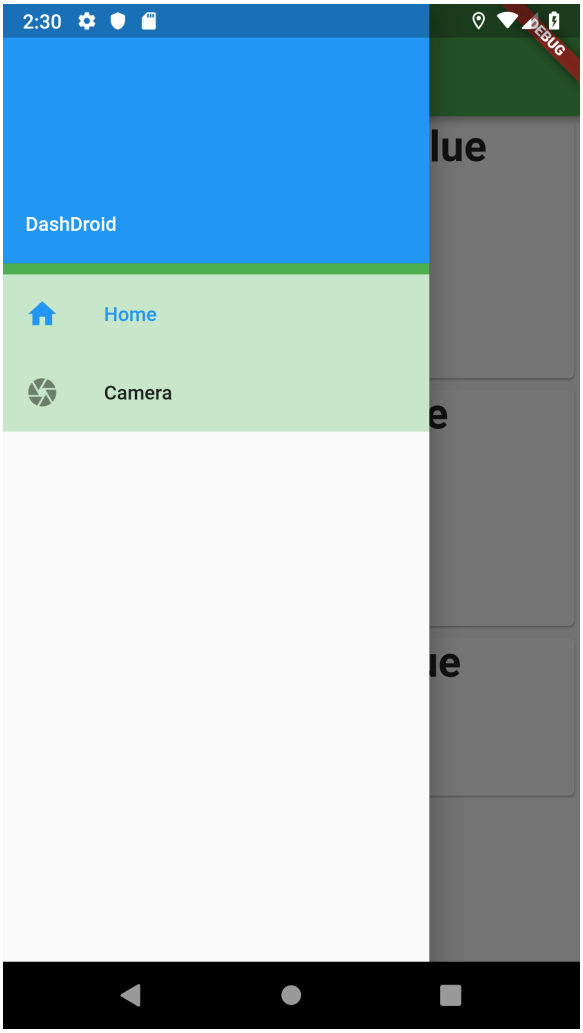

Figura 3: Drawer navigation Android

Da questo menu ci si può spostare sulla pagina della fotocamera o ritornare sulla home. Da notare il fatto che ora i due pulsanti di navigazione "Home" e "Camera" sono dentro la confort zone.

4.3 Camera IOS

Figura 4: Camera non attiva IOS

Nella figura 4 è presente l'altra pagina dell'applicazione ovvero la fotocamera. In questa pagina vi è, in alto, l'appbar con il nome della pagina in cui ci si trova ora, sotto l'appbar ci sono due *radio button* che permettono di selezionare la fotocamera con cui scattare, frontale o laterale. Al centro dello schermo c'è uno spazio dedicato alla fotocamera, inizialmente posto a nero con indicato di scegliere una fotocamera tra frontale e posteriore. Una volta selezionata la fotocamera frontale o posteriore (rispettivamente figura 5 e figura 6) è possibile scattare una foto con il pulsante rotondo in basso. Questo pulsante è posto vicino alla barra di navigazione per il fatto che generalmente la fotocamera ha il pulsante di scatto in basso e quindi per non disorientare l'utente ho deciso di posizionarlo in quel punto. Comunque il pulsante di scatto è grande abbastanza da poter essere premuto facilmente e senza il minimo errore. Una volta scattata, la fotografia viene memorizzata della cartella di debug di flutter per il semplice fatto che non è scopo di quest'applicazione salvare in un rullino le fotografie poichè si tratta di un'applicazione che mostra i vari sensori dello smartphone inclusa la fotocamera.

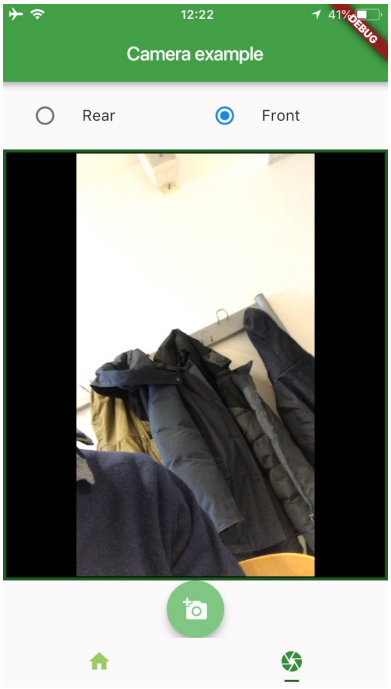

Figura 5: Fotocamera frontale IOS

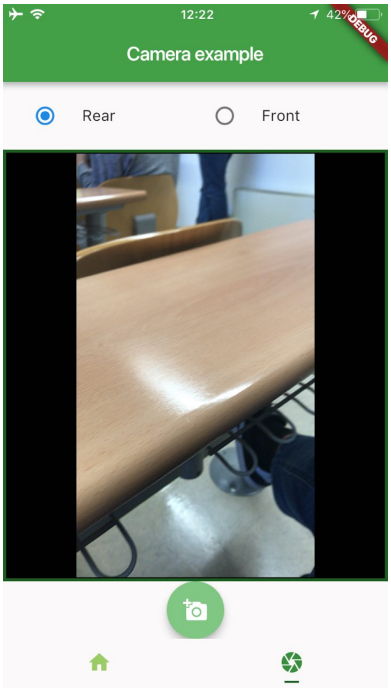

Figura 6: Fotocamera posteriore IOS

4.4 Camera Android

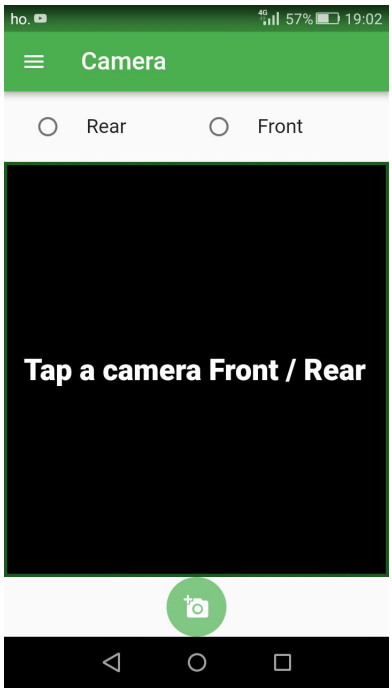

Figura 7: Camera non attiva Android

In figura 7 è presente la pagina della fotocamera su Andorid. E' sempre presente l'appbar con il nome della pagina in cui ci si trova per far si che l'utente non si senta disorientato nell'applicazione. Vi sono sempre i due radio button per selezionare la fotocamera frontale o laterale ed è presente il bottone di scatto in basso, questo pulsante è comunque abbastanza grande da non far commettere errori all'utente essendo vicino alla barra di navigazione del sistema operativo. In figura 8 si vede la fotocamera posteriore attiva.

Figura 8: Fotocamera posteriore IOS

5 Considerazioni su Flutter

In questa sezione riporto un breve commento sugli aspetti positivi e negativi incontrati nello sviluppo dell'applicazione *DashdroidFlutter*.

5.1 Aspetti positivi

Un fatto sicuramente positivo è che con Flutter è possibile realizzare un'applicazione dal design moderno in veramente pochi passaggi, non si trova alcuna difficoltà ad imparare le meccaniche iniziali e il posizionamento dei widgets nell'interfaccia. La documentazione è veramente ben fatta ed accurata, con vari esempi a seconda di quale macro-tipo di applicazione di vuole realizzare. Sono disponibili inoltre varie librerie da utilizzare con Dart per accedere alle funzionalità basilari del proprio device.

5.2 Aspetti negativi

Purtroppo, nel caso della realizzazione dell'applicazione in questione, devo ammettere che Flutter non è il framework giusto perché le librerie che ho trovato per l'accesso ai sensori sono state solamente quelle per la fotocamera, accelerometro, giroscopio e GPS. Inoltre ho avuto difficoltà con la libreria per il GPS in quanto la versione disponibile era datata e funzionante con la versione di Dart 1. Ho avuto inoltre difficoltà con la manipolazione degli *Streams* per leggere i valori dei sensori, infatti viene letto il valore dall'accelerometro o giroscopio in tempo reale e aggiornato ad ogni minima variazione; ciò è utile nel caso si dovesse utilizzare tale valore per operare azioni sulla vista per esempio ruotare lo schermo o ruotare un oggetto all'interno dello schermo, ma per leggere i valori e mostrarli all'utente si incontrano varie difficoltà per il semplice fatto che l'aggiornamento dei valori avviene in maniera troppo veloce. Ho tentato di rallentare tale aggiornamento cercando di manipolare lo stream ma non sono riuscito anche se suppongo si possa fare.

Conclusioni

Quindi alla luce dei pregi e difetti riportati sopra, considero Flutter un ottimo framework per la realizzazione di applicazioni con design piacevole in tempi molto rapidi. Invece per l'utilizzo dei sensori e più in generale per la creazione di applicazioni più complesse ritengo che Flutter sia ancora immaturo poichè mancano ancora varie librerie disponibili pubblicamente e dovendo scrivere codice nativo per compensare tale mancanza, riduce l'utilità di utilizzare un framewrok cross-platform.

6 DashDroid Xamarin

Questa sezione elenca e descrive le principali scelte implementative dell'applicazione Dashdroid realizzata con il framework cross-platform Xamarin.forms; il codice completo è visibile al repositori di github: `https://github.com/P4rTY6/XamarinDashdroid.git` .

6.1 Home page IOS

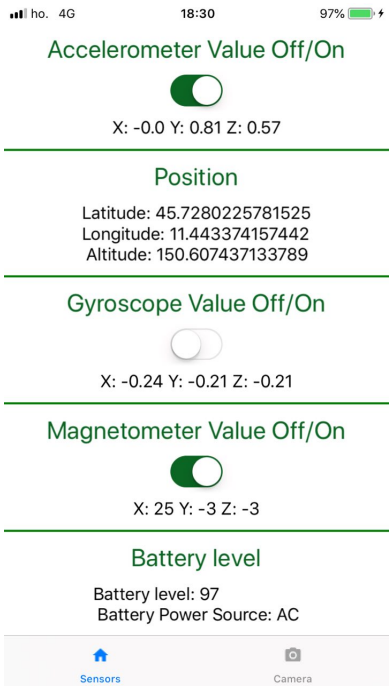

Figura 9: Home page IOS

Nella figura 9 si può vedere la home page su IOS, nella parte centrale vi sono 5 view separate da una riga orizzontale. Ogni view contiene il titolo indicante il sensore da cui vengono ottenuti i dati, uno *switch button* che, inizialmente disattivato, permette di attivare la lettura continua dei valori dallo specifico sensore. I bottoni seguono la logica comune in cui sono attivati quando il pallino centrale è spostato verso destra e il colore di riempimento è attivo (verde in questo caso), mentre quando sono disattivati il pallino è verso sinistra e il contenuto è offuscato e di colore grigio sbiadito. Quindi l'utente essendo abituato a questa convenzione, con i bottoni switch, non rischia di essere disorientato. Infine, nella view, vi è il valore letto dal sensore, questo valore è letto alla velocità UI cioè una velocità adatta per l'interfaccia utente, cosa che Xamarin permette di fare diversamente da Flutter. Se viene disattivata la lettura dal sensore, attraverso lo switch button, il valore letto rimane bloccato all'ultimo valore letto, facendo in modo che l'utente possa visualizzarlo con più calma. Se un sensore è mancante oppure non funziona correttamente allora viene restituito l'alert (in figura 11). In fondo alla pagina troviamo la *bottom tab navigation* in cui ci si può spostare tra le due pagine dell'applicazione (home e camera), inoltre sono abilitate le gesture in quanto attraverso uno swipe si può navigare tra l'applicazione. La bottom tab navigation, su IOS, è in fondo alla pagina nella comfort zone secondo le comuni indicazioni sul design UX per IOS.

6.2 Home page Android

La figura 10 mostra la home page su Android in cui valgono gli stessi discorsi fatti per la home page su IOS a parte la barra di navigazione che su Android è posta in alto per il fatto che è presente la barra del sistema operativo in basso e non è opportuno posizionare due barre di navigazione una sopra l'altra. Anche su Android sono supportate le gestures, in particolare lo swipe per navigare tra le due pagine dell'applicazione.

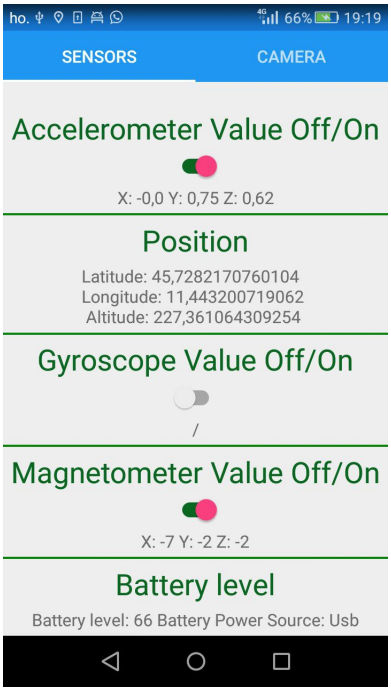

Figura 10: Home page Android

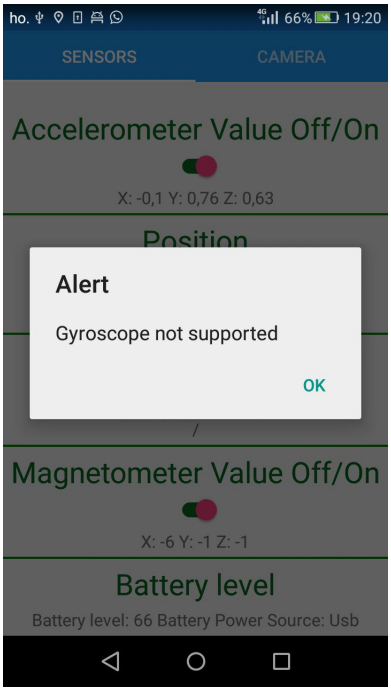

Figura 11: Alert sensore non supportato

In figura 11 si può vedere come vengono gestiti gli errori nel caso il sensore non sia presente o non funzioni correttamente. Questo alert viene mostrato solamente quando si tenta di attivare il sensore attraverso lo switch button.

Figura 12: Fotocamera su IOS

6.3 Fotocamera IOS

Nella figura 12 si può vedere l'altra pagina dell'applicazione dove è possibile scattare una foto e visualizzarla in questa schermata. La pagina mostra al centro la foto appena scattata altrimenti ha uno spazio bianco. Sotto è presente un bottone che se premuto apre la fotocamera del dispositivo e, una volta scattata la foto, chiede se la si vuole salvare oppure scartare e rifarne un'altra, nel caso la si voglia tenere viene chiusa la fotocamera e si ritorna alla pagina dell'applicazione che ora avrà al centro la fotografia appena scattata. In basso è sempre presente la barra di navigazione indicante che ora ci troviamo nella pagina *camera*, ben distanziata dal bottone *Take Photo*.

6.4 Camera Android

In figura 13 è presente la pagina della fotocamera su Andorid. Valgono le stesse considerazioni fatte sulla pagina della fotocamera su IOS a parte il fatto che la barra di navigazione su Android è posta in alto per stare separata dalla barra del sistema operativo.

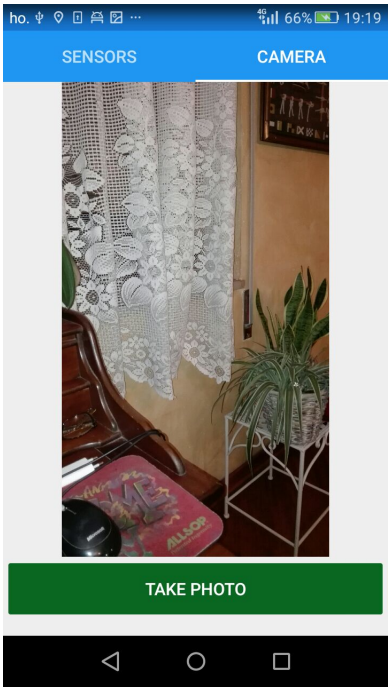

Figura 13: Fotocamera su Android

7 Considerazioni su Xamarin.Forms e confronto con Flutter

In questa sezione riporto un breve commento sugli aspetti positivi e negativi incontrati nello sviluppo dell'applicazione *Dashdroid* con Xamarin.Forms confrontandoli con l'esperienza avuta nello sviluppo con Flutter.

7.1 Aspetti positivi di Xamarin

Nello sviluppo dell'applicazione Dashdroid con il framework Xamarin.Forms mi sono trovato molto bene in quanto ho trovato molto materiale di studio, oltre le slide a lezione, in quanto il framework è disponibile da molto tempo e ha una vasta community a supporto. Anche per quanto riguarda il modo di organizzare l'architettura dell'applicazione, con il pattern MVVM e utilizzando XAML per la creazione della user interface e C# per il controllo sulla view, è molto intuitivo e consente di organizzare meglio la propria applicazione e scrivere codice manutenibile e chiaro. Per creare un'applicazione cruscotto, che legga i valori dai sensori dello smartphone, Xamarin.Forms si è rivelato un framework molto adatto, molto di più di Flutter. Infatti sono disponibili varie librerie (Xamarin.Essential) per leggere i valori dei sensori, molti di più rispetto a quelli con Flutter, e inoltre non sono stati trovati bug o problemi nell'uso di queste librerie esterne, cosa che era successa con Flutter.

7.2 Aspetti negativi di Xamarin

Nonostante venga messo a disposizione una preview grafica dell'interfaccia realizzata con XAML non è presente l'hot-reloading, quindi per testare l'applicazione su un emulatore o sul proprio dispositivo, ci vuole del tempo per compilare il codice e quindi alla lunga si perde molto tempo. Mentre su Flutter era disponibile l'hot-reloading, quindi la fase di test sul dispositivo è stata molto più veloce. Un problema è il fatto che xamarin, di base, non ha un design moderno come il material design di Flutter ma è sufficiente lavorare sullo stile per ottenere buoni risultati.

Conclusioni

Considerando i pregi e difetti dei due framework posso concludere che Xamarin.Forms è stata la miglior scelta per realizzare l'applicazione cruscotto per leggere valori dai sensori, sia per la facilità di utilizzo delle API messe a disposizione sia per la maturità del framework che ha una vasta community a supporto e vari tutorial su come iniziare a sviluppare. Per lo sviluppo di un prototipo di un'applicazione in poco tempo, sceglierei Xamarin.Forms in quanto anche se il layout non sarà moderno e accattivante al pari di quello ottenuto utilizzando Flutter comunque

posso creare un'applicazione molto più velocemente e più completa avendo a disposizione molte più librerie e non dovendo ricorrere al nativo perdendo l'utilità di usare un framework cross-platform.