ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

PROGETTAZIONE E SVILUPPO MULTI-PLATFORM DI APPLICAZIONI MOBILE: UN CASO DI STUDIO IN XAMARIN

Elaborato in
PROGRAMMAZIONE DI SISTEMI EMBEDDED

Relatore
Prof. ALESSANDRO RICCI

Co-relatore
Ing. ANGELO CROATTI

Presentata da
ENRICO CECCOLINI

Terza Sessione di Laurea
Anno Accademico 2014 – 2015

PAROLE CHIAVE

Multi-platform

Mobile computing

Xamarin

Alla mia famiglia, a Sara e ai miei amici.
Ai miei professori dell'ITIS di Urbino che mi hanno avvicinato
a questa materia.

Indice

Introduzione

La forte diffusione dei dispositivi mobile, smartphone e tablet, ha fatto in modo che, nello stato attuale, il tempo che gli utenti trascorrono online abbia raggiunto il 50% tramite le applicazioni mobile. Questo, unito alla tendenza che vede sempre più persone abbandonare completamente l'utilizzo del computer, obbliga le aziende fruitrici di servizi online a rendere disponibile il loro accesso anche tramite app, per poter sperare di essere competitivi sul mercato.
La frammentazione del mercato dei sistemi operativi mobile, crea problemi nel raggiungimento di un'audience totale per i propri servizi, e, il suo andamento instabile, può portare a cattivi investimenti da parte di aziende che si trovano a dover decidere per quali piattaforme fornire la propria applicazione. I framework per lo sviluppo multi-platform di applicazioni mobile, intervengono per coadiuvare lo sviluppatore nella non specializzazione del processo di progettazione e sviluppo per le singole piattaforme.
L'impiego di questi framework può essere utile in previsione di una necessità futura di supporto di altre piattaforme oltre a quelle inizialmente selezionate e anche per ridurre il costo di produzione e manutenzione dell'applicazione, creando un ambiente centralizzato per la codifica. Ne esistono molti, ognuno corrispondente ad un approccio multi-platform tra i seguenti: approccio web, approccio ibrido e approccio cross-compilato. Quest'ultimi saranno analizzati assieme ai relativi framework per poter essere confrontati, andando a costituire una base decisionale sulla quale lo sviluppatore può individuare quale adottare, a seconda delle situazioni e delle proprie esigenze.
L'approccio ritenuto da me più interessante sarà approfondito e valutato tramite lo sviluppo di un caso di studio reale.
La tesi è suddivisa in questo modo:

- nel primo capitolo vengono identificati i motivi che evidenziano la necessità di adottare un approccio multi-platform, chiarendo di che cosa si tratta e lo stato dell'arte degli approcci e dei relativi framework, concludendo con un confronto;
- nel secondo capitolo viene svolto un approfondimento su Xamarin, il framework da me individuato come più interessante al momento, definendo

funzionamento interno e strumenti messi a disposizione;

- nel terzo capitolo viene presentato il caso di studio, andando a definire il problema e il perché può essere considerato per valutare l'approccio e il framework selezionato;
- nel quarto capitolo viene mostrato il processo di progettazione, sviluppo e test dell'applicazione, evidenziando le differenze rispetto all'utilizzo di un approccio nativo, per poi trarre le conclusioni del lavoro svolto.

Capitolo 1

Stato dell'arte dello sviluppo multi-platform

Al primo posto tra le innovazioni tecnologiche che hanno avuto più impatto sulle nostre vite negli ultimi tempi, troviamo senza dubbio lo smartphone, idea che, insieme a quelle che ora tutti conoscono come app, si trova al terzo posto tra le più rilevanti degli ultimi 30 anni[1]. L'avvento di questa tipologia di dispositivo mobile è stato accolto dal pubblico in maniera talmente entusiasta da rendere la sua crescita evolutiva molto rapida, con la partecipazione di numerose aziende produttrici di hardware e software. Uno studio significativo che certifica l'interesse dei consumatori per tali dispositivi, calcola che nel 2018 più del 50% degli utenti internet utilizzerà un tablet o uno smartphone come unico strumento per accedere a vari servizi online[2], mentre già dal 2014 più del 50% del tempo che gli utenti passano online avviene tramite l'utilizzo di app[3] (web browser del telefono escluso).

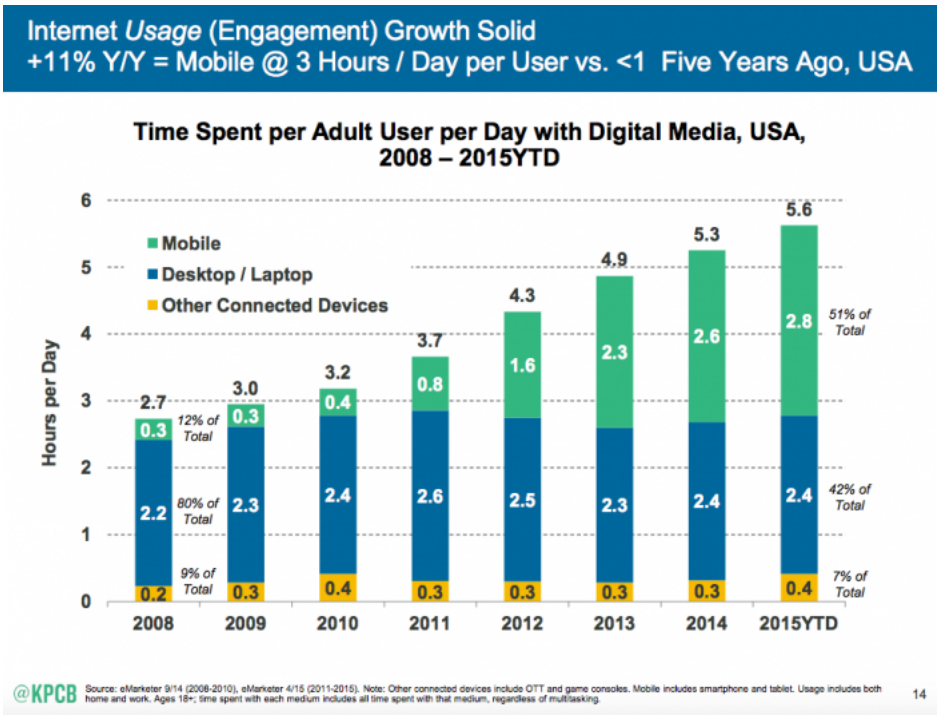


Figura 1.1: Utilizzo di Internet dal 2008 al 2015[4]

Mentre un tempo il motivo che spingeva un'azienda a commissionare la realizzazione di una un'app per il proprio business, poteva essere quello di contribuire a dare un'immagine innovativa al proprio marchio, ora la presenza nel mondo mobile è diventata un prerequisito importante per poter essere competitivi nel mercato. Gli utilizzatori di smartphone sono oramai abituati alla pratica di cercare negli store della propria piattaforma l'app che li può aiutare a soddisfare i propri bisogni, prima ancora di una ricerca su browser, e, il non essere presenti, crea una perdita significativa di clienti. Ad esempio: un' azienda che possiede un servizio di vendita online tramite un sito internet, non può ignorare che, secondo lo studio sul fenomeno di diffusione delle tecnologie mobile, l'e-commerce tramite smartphone e tablet raggiunga il 50% del totale entro il 2018[5]. Chiarita l'importanza per le aziende di rendere disponibili i propri servizi per i dispositivi mobili, occorre specificare che un potenziale raggiungimento della totalità dei dispositivi ora in commercio è pressoché impossibile. Il mercato dei sistemi operativi per smartphone si presenta fortemente frammentato in termini di offerta e, date le differenze sostanziali tra i vari SO, prendere in considerazione tutte le piattaforme può equivalere al dover realizzare una versione della propria applicazione per ogni piattaforma a cui si vuole fornire supporto. I principali fornitori di sistemi operativi in termini di volumi di vendita sono quattro, di cui due detengono una quota del mercato

che si attesta oltre al 96%: Android di proprietà di Google con l'82.8% e iOS di Apple con il 13.9%. Gli altri sono Windows Phone di Microsoft con il 2.6% e BlackBerry OS con lo 0.3%.

Period	Android	iOS	Windows Phone	BlackBerry OS	Others
2015Q2	82.8%	13.9%	2.6%	0.3%	0.4%
2014Q2	84.8%	11.6%	2.5%	0.5%	0.7%
2013Q2	79.8%	12.9%	3.4%	2.8%	1.2%
2012Q2	69.3%	16.6%	3.1%	4.9%	6.1%

Figura 1.2: Quote di mercato dal 2012 al 2015[7]

La tabella in figura 1.2 mostra le quote di mercato delle varie piattaforme dal 2012 al 2015, ma, oltre a certificare il dato per cui si può raggiungere un audience del 99.3% fornendo supporto per i sistemi operativi più diffusi, mette anche in guardia sull'inclinazione dello stesso a forti variazioni. Allontanando la lente di ingrandimento otteniamo la conferma di tale andamento [Figura 1.2].

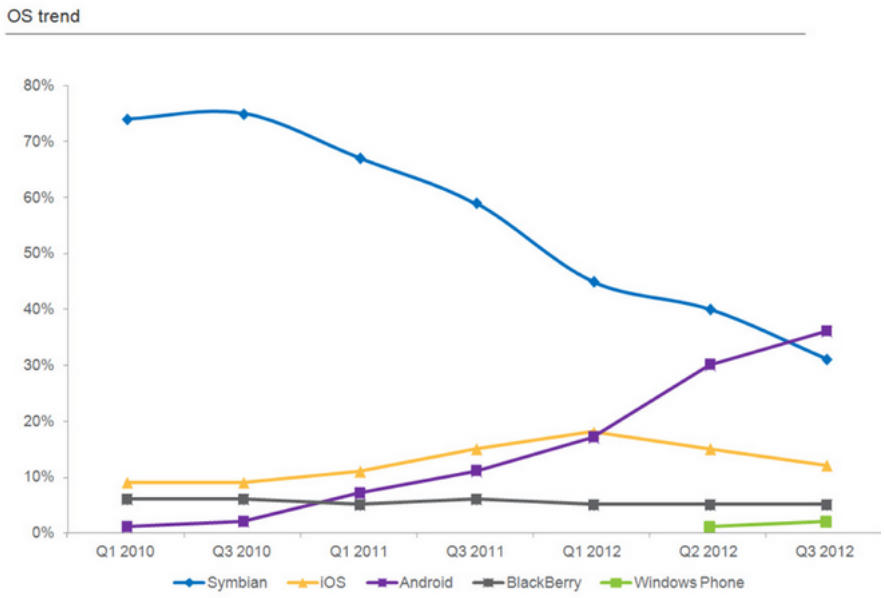


Figura 1.3: Quote di mercato dal 2010 al 2012[8]

Lo sviluppo platform-specific diventa una pratica non solo dispendiosa, dovendo prevedere l'intervento di diverse persone o team che possiedono conoscenze e abilità diverse, ma anche inefficiente per rispondere in maniera rapida a eventuali variazioni del mercato come ad esempio l'ascesa di un sistema operativo.

Un approccio alternativo che può essere di aiuto per affrontare i problemi individuati è quello dello sviluppo multi-platform. Cercheremo di capire in seguito quali diversi approcci raggruppa e in che modo può rappresentare una valida soluzione.

1.1 Sviluppo multi-platform

Lo sviluppo multi-platform, in contrasto con quello che è lo sviluppo platform-specific, consiste nell'adottare un approccio che permetta allo sviluppatore di utilizzare codice sorgente comune nella creazione di una applicazione che ha come target piattaforme diverse. Tale idea risulta essere di indubbio fascino dal punto di vista ingegneristico poiché, già dalla sua definizione, introduce aspetti di grande rilevanza su quella che è la qualità di un progetto software. In maniera diretta troviamo infatti la caratteristica di **portabilità**, cioè la capacità di funzionare su più piattaforme. Indirettamente si possono invece ottenere vantaggi per quanto riguarda:

- **Produttività** (l'efficienza del processo di produzione in termini di velocità di consegna): lo sviluppo dell'applicazione conterrà elementi in comune tra le piattaforme anche in fase di realizzazione, oltre che nella sua fase di analisi e progettazione, essendo previsto l'utilizzo degli stessi strumenti per le diverse piattaforme target;
- **Facilità di manutenzione**: le eventuali modifiche correttive, adattive e di aggiunta di funzionalità, saranno effettuate in un'unica soluzione.

Detto questo, occorre guardare quanto e in che modo questa scelta influisca sulle altre qualità come: efficienza, facilità d'uso e affidabilità. Questo dipende sia dall'approccio multi-platform, sia dalla capacità dello strumento adottato di far fronte alle differenze sostanziali tra i sistemi operativi target; meglio queste verranno affrontate, più il metodo sarà buono. E' necessario descrivere le principali differenze in modo da poter capire quali sono i problemi che potrebbero sorgere quando si sviluppa del codice multi-platform. Le principali differenze sono:

- **Linguaggio di programmazione**: le applicazioni platform-specific sono scritte e compilate utilizzando il linguaggio di programmazione proprio del sistema operativo e delle librerie proprietarie.

Piattaforma	**Linguaggi utilizzati**
iOS	Objective-C, Swift e C
Android	Java, C e XML
Windows Phone	C#, C++, VisualBasic e XAML
BlackBerry	Java e C++
Tizen	C++ e C
Ubuntu Touch	QML, C e C++
Firefox OS	HTML, CSS e JavaScript

E' chiaro che, essendo il suo obiettivo primario la condivisione di codice, un approccio multi-platform deve fare riferimento il meno possibile ai linguaggi nativi in favore di uno in comune;

- **Ambiente di sviluppo e Tools**: per realizzare un'applicazione platform-specific occorre aver installato nel proprio computer l'IDE (Integrated Development Environment) e l'SDK (Software Development Kit) propri della piattaforma. Questo significa, nella maggior parte dei casi, il dover utilizzare uno specifico sistema operativo, unico in grado di far girare i vari strumenti.

Piattaforma	**IDE e Sistema Operativo**
iOS	Xcode su MAC con OS aggiornato
Android	Android Studio, Eclipse o Netbeans
Windows Phone	Visual Studio su Windows con OS aggiornato
BlackBerry	BackBerry Java Eclipse Plug-in
Tizen	Tizen IDE
Ubuntu Touch	QT Creator su Ubuntu
Firefox OS	Firefox WebIDE

E' sicuramente un altro aspetto che va accomunato il più possibile, lasciando al massimo l'operazione di compilazione e di esecuzione agli specifici sistemi;

- **Interfaccia utente**: un importante ostacolo alla programmazione multi-platform è rappresentato dalla diversità delle interfacce utente che portano ad una esperienza differente in ogni piattaforma. Nella maggior parte delle situazioni è fondamentale ottenere una esperienza d'uso nativa per garantire la facilità di utilizzo da parte di utenti abituati ad utilizzare il sistema da loro scelto;

- **Supporto al multitasking**: il multitasking, servizio che permette di mantenere attive applicazioni in background in favore di altre, è realizzato in modo differente tra i diversi sistemi operativi. I framework di sviluppo multi-platform devono fornire strumenti unici in grado di adattarsi al meglio alla piattaforma sottostante;

- **Pubblicazione dell'applicazione**: ogni piattaforma possiede un proprio Store ufficiale attraverso il quale distribuire le proprie applicazioni. Requisito fondamentale per poter pubblicare la propria applicazione su uno specifico store, è l'essere in possesso dell'archivio binario dell'app(file eseguibile) appropriato.

Piattaforma	**Store, formato file eseguibile**
iOS	Apple iTunes, .app
Android	Android Market, .apk
Windows Phone	Microsoft Store, .xap o .appx
BlackBerry	BackBerry App World, .cod
Tizen	Tizen Store, .tpk
Ubuntu Touch	Ubuntu Software Centre, .deb
Firefox OS	Firefox Marketplace, .zip contentente i sorgenti

  Il set di strumenti nativi ne permette l'ottenimento in modo semplice, i framework di sviluppo multi-platform devono provvedere ad una soluzione nel miglior modo possibile.

La famiglia degli approcci multi-platform è molto ampia, per questo motivo andremo ad analizzare i suoi componenti uno ad uno indicando l'idea di base e le conseguenti caratteristiche positive e negative.

1.1.1 Approccio web, le web app

Le Web app sono applicazioni progettate per essere eseguite tramite l'utilizzo del browser-web installato sul dispositivo, in modo tale da ottenere la portabilità tra le varie piattaforme. Queste non risiedono direttamente sui dispositivi, ma su server che le rendono accessibili tramite Internet fornendo un indirizzo URL. La realizzazione prevede l'acquisizione di uno spazio web che può avere costi più o meno elevati in base alla complessità dell'applicazione e il numero di utilizzatori. Tali applicazioni presentano un'architettura client-server dove il front-end, corrispondente all'interfaccia grafica, deve essere rigorosamente implementato tramite l'utilizzo di linguaggi web interpretabili dal browser come HTML, CSS e JavaScript, mentre per il back-end si ha libertà di scelta. Non sarà possibile ottenere un look&feel nativo a seconda della piattaforma utilizzata. Gli utilizzatori non avranno bisogno di installare nulla nel proprio dispositivo, saranno sottoposti ai vari aggiornamenti in maniera immediata e totalmente automatica tramite un nuovo accesso all'applicazione. Essendo previsto un server web si ha un notevole risparmio di risorse, la maggior parte della computazione non viene svolta localmente, si ottiene così un potenziale aumento di capacità di calcolo rispetto a quella fornita dal dispositivo ospite. Le web app non riescono però a soddisfare diverse richieste che nella maggior parte dei casi sono indispensabili per poter essere competitive sul mercato. Ad esempio, come rovescio della medaglia all'assenza di installazione abbiamo la loro impossibilità di essere presenti negli store delle varie piattaforme, rendendole difficili da recuperare. Con la presenza dell'agente server nella struttura vera e propria dell'applicazione, possono incorrere in problemi di robustezza in caso di non accesso ad internet o assenza di servizio. Altro grave difetto è la difficoltà, che spesso diventa impossibilità, di accesso alle risorse del dispositivo, non essendo mai diretto ma sempre tramite browser.

1.1.2 Approccio ibrido, le app ibride

L'approccio ibrido si colloca a metà tra quello nativo e quello web. Un' applicazione ibrida viene realizzata tramite l'utilizzo di tecnologie web ma, a differenza delle web app, viene eseguita in un wrapper nativo che funge da contenitore. La presenza di tale elemento scritto in liguaggio nativo permette la realizzazione di un layer astratto che a sua volta rende possibile un mapping delle funzionalità del dispositivo (GPS, fotocamera, rubrica, sistema notifiche,...) in modo da renderle disponibili agli sviluppatori tramite l'utilizzo di API JavaScript. La disponibilità o meno di una funzionalità nativa dipende fortemente dalle scelte degli sviluppatori del framework adottato. Ad esempio: in uno scenario di uscita di una nuova versione del sistema operativo, per po-

ter sfruttare le caratteristiche introdotte, occorrerà attendere l'aggiornamento del framework. Una differenza sostanziale tra i diversi framework di sviluppo ibrido è la capacità o meno di utilizzare elementi dell'user interface nativa. Se questa funzionalità è mancante il rendering grafico viene affidato ad una *WebView*, un componente presente nel SDK dei vari sistemi. Tali WebView nascono con l'idea di supportare le applicazioni native per mostrare codice HTML ottenuto da un server remoto ma, il wrapper fornito dal framework scelto, rende invece possibile la visualizzazione delle pagine delle di una applicazione ibrida appena questa viene lanciata, rendendo possibile la realizzazione di una vera e propria UI. Tramite l'utilizzo di appositi tool grafici è possibile simulare il look&feel nativo ottenendo gradevoli risultati.

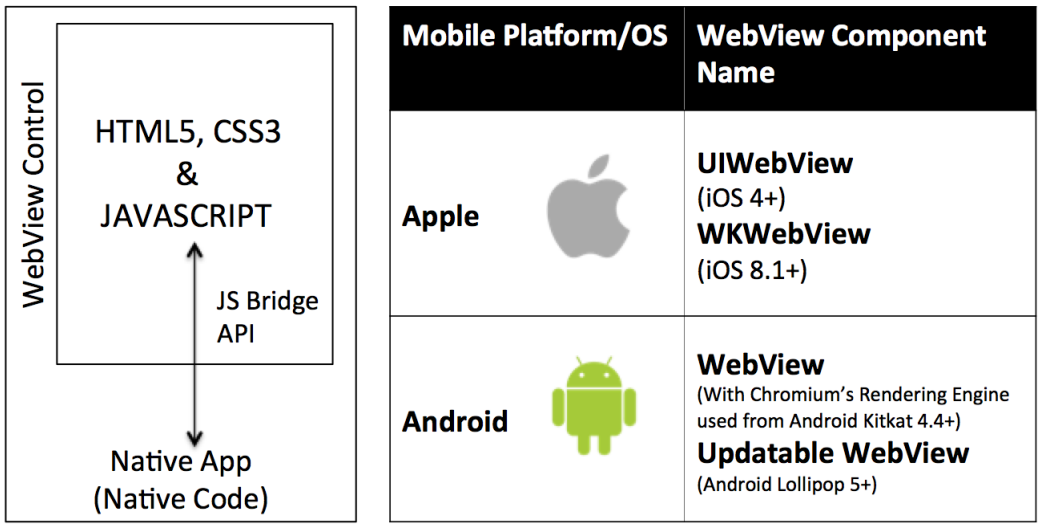


Figura 1.4: WebView[9]

Nel caso in cui il framework permetta di utilizzare l'UI nativa non sarà necessario l'utilizzo della WebView del sistema per la visualizzazione dei componenti grafici, ma, lo sviluppatore, potrà utilizzare le specifiche API per ogni piattaforma a cui si vuole dare supporto garantendo una esperienza di utilizzo nativa a discapito di un totale riutilizzo del codice. In entrambi i casi, al momento della compilazione il framework si occuperà di fare il deploy dell'applicazione sulle varie piattaforme supportate, ottenendo i vari file di installazione. Per questo motivo, a differenza delle web app, le app ibride possono essere inserite sugli store assicurandosi un pubblico maggiore.

Tra i framework più utilizzati troviamo:

- **Apache Cordova**: Apache Cordova è un ambiente di sviluppo open-source per la creazione di applicazioni mobile ibride tramite l'utilizzo del-

le WebView native. Le piattaforme attualmente supportate sono: iOS, Android, Windows Phone, BlackBerry, Amazon-fireOS e in parte Firefox OS e Ubuntu. La popolarità di Apache Cordova deriva dalla qualità degli strumenti messi a disposizione agli sviluppatori, in pochi passi è infatti possibile costruirsi un base completa sulla quale andare in seguito a sviluppare la propria applicazione. Con l'utilizzo del tool *Cordova CLI*, presente per tutti i principali sistemi operativi desktop, tale processo è composto da questi step:

- creazione applicazione dandogli un nome;
- aggiunta delle piattaforme da supportare;
- aggiunta dei Cordova plugin necessari.

Un Cordova plugin corrisponde a del codice che fornisce un'interfaccia JavaScript verso un componente nativo. Ne esistono più di mille[10], tutti ottenibili tramite Cordova CLI. Per ognuno di questi è disponibile una documentazione ricca di indicazioni ed esempi che vengono principalmente dalla community che si è creata. Ricapitolando: al termine del processo elencato in precedenza, abbiamo Apache Cordova che fornisce un'app vuota per ogni piattaforma, da riempire con codice HTML5, CSS3 e JavaScript comune. In seguito, Cordova si occuperà di creare i relativi archivi binari da poter distribuire sugli store. Per quanto riguarda la creazione dell'interfaccia grafica è possibile ottenere un look&feel simil-nativo tramite l'utilizzo di framework grafici come Ionic, Famo.us, Framework7 e OnsenUI che si occupano di creare delle versioni web di tutti gli elementi grafici nativi.

- **Titanium**: Titanium è un ambiente di sviluppo per la creazione di applicazioni mobile ibride e web. Le piattaforme supportate sono iOS, Android e Windows Phone. L'obiettivo di Titanium, come riportato nella sua documentazione, è quello di fornire agli sviluppatori uno strumento che elimini quanto più è possibile quelli che sono i compromessi dello sviluppo ibrido. Le applicazioni create con Titanium sono costruite interamente in JavaScript. Il framework mette a disposizione sia API javaScript platform-specific che cross-platform per costruire applicazioni con look&feel nativo e prestazioni il più possibile vicine a quelle native. Titanium implementa classi proprietarie per offrire funzionalità del device e componenti di UI nativi tramite API JavaScript. Tutto il codice sorgente dell'app, scritto in JavaScript, viene "inglobato" in un file nativo per poter essere distribuito. L'applicazione viene eseguita dentro un interprete JavaScript incorporato nel codice sorgente durante la compilazione. Tale interprete valuta il codice a runtime, facendolo interagire

con l'ambiente nativo attraverso oggetti proxy che esistono su tutti gli ambienti supportati.

1.1.3 Approccio cross-compilato

Nell'approccio cross-compilato si ha che le applicazioni vengono scritte in un linguaggio di programmazione comune e un cross-compiler si occupa della traduzione del codice sorgente in file binari nativi. Si evidenzia dunque l'importanza dell'efficienza e dell'affidabilità di tale compilatore visto che l'operazione di cross-compilazione di parti di codice particolarmente articolato è tutt'altro che immediato. L'enorme vantaggio di questo approccio è la possibilità di accedere, quando è necessario, a tutte le caratteristiche native della piattaforma e tutti i componenti dell'interfaccia grafica nativa possono essere utilizzati senza difficoltà. Il problema è che l'interfaccia grafica non può essere riutilizzata, così come il codice riguardante le caratteristiche hardware native utilizzate. Queste caratteristiche dipendono infatti dalla specifica piattaforma e il modo di accedervi e strettamente legato ad essa.

Tra i framework più utilizzati troviamo:

- **Xamarin**: Xamarin è un ambiente di sviluppo per la creazione di applicazioni mobile che ha avuto un grande successo negli ultimi anni e che continua a guadagnare sempre più interesse. Xamarin fa uso di un wrapper C# in grado di raggiungere tutte le API native di ogni piattaforma, permettendo agli sviluppatori di interagire con quest'ultima nello stesso modo di uno sviluppatore di codice nativo. Per eseguire il codice sul dispositivo, Xamarin si appoggia a Mono, un'implementazione open source del framework Microsoft .NET, che si affianca alla macchina virtuale nativa del sistema operativo, permettendo dunque il code sharing tra piattaforme.

1.2 Considerazioni finali sugli approcci

Per un miglior confronto tra i vari approcci, riepilogo quanto visto in precedenza nelle seguenti tabelle:

	Nativo	**Web**	**Ibrido**	**Cross-compilazione**
Tempi e costi di sviluppo	Alti	Bassi	Medi	Medi
Conoscenza piattaforme	Alta, conoscenza dei vari linguaggi nativi necessaria	Bassa	Media, conoscenza aspetti piattaforma necessaria	Media, conoscenza aspetti piattaforma necessaria
Utilizzo offline	Si	No	Si	Si
Modalità di distribuzione	Store o file installazione	Solo Web	Store o file installazione	Store o file installazione
Integrazione col device	Completa	Molto limitata	Buona	Completa
Performance	Ottime	Dipendente dalla connessione e dal web server	Buone	Ottime
Look&feel	Nativo	Nativo simulato	Nativo o simil-nativo, dipendende dal framework scelto	Nativo
Portabilità	Nessuna	Ottima	Ottima, dipendente dal framework scelto	Ottima, dipendente dal framework scelto

Tabella riepilogativa degli approcci multi-platform

	Ibrido		**Cross-compilato**
	Apache Cordova	**Titanium**	**Xamarin**
Linguaggi	JavaScript, HTML, CSS	JavaScript, HTML, CSS	C#, XAML
supporto iOS, Android, Windows Phone	Si	Si	Si
supporto BlackBerry, AmazonfireOS, Firefox OS, Ubuntu	Si	No	No
user interface nativa	No	Si	Si

Tabella riepilogativa dei principali framework multi-platform

La scelta dell'approccio e del framework da utilizzare per lo sviluppo di una applicazione è un aspetto molto importante per la buona riuscita del progetto. Per questo è fondamentale che avvenga solo dopo aver fatto chiarezza su seguenti punti:

- I requisiti funzionali del progetto;
- L'evoluzione che potrebbe avere il progetto;
- I sistemi operativi mobile da supportare;
- Le competenze delle persone che andranno a sviluppare l'applicazione;
- Il tempo utile per disporre della soluzione;
- Il budget a disposizione.

Visto che i vari approcci mancano di funzionalità ben definite, è quasi sempre possibile arrivare alla soluzione ottima procedendo per esclusione. Le tabelle presentate possono aiutare in questo processo rispondendo alle domande che rappresentano i bisogni dello sviluppatore come:

- Ho la necessità di distribuire l'app tramite gli store?
- Ho la necessità di un utilizzo offline?

- E' necessario integrarsi con i sensori del dispositivo?
- Il *time-to-market* è fondamentale?
- Necessito di un'user interface nativa?
- Necessito di animazioni ed effetti?
- Dove trovo supporto per i miei sistemi operativi?
- Dove posso utilizzare i linguaggi che conosco?

Come si può vedere dalle domande e dalle relative risposte, non sempre si arriva ad avere come risultato ottimo una soluzione multi-platform. Sceglierla per tutti i progetti, senza fare uno studio adeguato, è un approccio sbagliato.

Come ultima considerazione sulla scelta della tecnologia da utilizzare per lo sviluppo di applicazioni mobile, va detto che, per progetti di grandi dimensioni, una soluzione che vede l'utilizzo di linguaggi che non supportano direttamente una programmazione orientata ad oggetti come JavaScript per la definizione della business logic, può essere tragica. Vista la mancanza di tutti quei meccanismi propri dell'object-oriented, l'impatto sulla qualità del software prodotto è quasi sempre notevole. Questo problema ha portato varie aziende, tra cui Google con il suo **Dart**[11] e Microsoft con il suo **TypeScript**[12], a sviluppare nuovi linguaggi orientati ad oggetti che fanno da *Super-Set* a JavaScript. Tuttavia, trattandosi di nuovi linguaggi, l'integrazione con i framework multi-platform esistenti non è ancora avvenuta, lasciando tale interessante discussione solo per sviluppi futuri. Per questo motivo è stato ritenuto di particolare interesse approfondire il framework Xamarin, che fa utilizzo di C#, linguaggio orientato ad oggetti in modo nativo con anni di esperienza.

Capitolo 2

Il framework Xamarin

In questo capitolo viene svolto un approfondimento su Xamarin, il framework da me individuato come più interessante al momento, definendo storia, funzionamento interno e strumenti messi a disposizione.

2.1 Storia

Poco dopo l'annuncio del framework .NET da parte di Microsoft, Miguel de Icaza e Friedman (fondatori di Ximian) iniziarono lo sviluppo di un progetto open-source chiamato Mono. L'idea era quella di creare una versione alternativa del framework .NET per sistemi Linux. Nel 2011 nasce Xamarin, azienda che pianificò la realizzazione di un ambiente di sviluppo per la creazione di applicazioni mobile che abbia come base Mono.

Mono è una implementazione cross-platform open source del framework .NET. Questa include il Common Language Runtime(CLR) cioè l'ambiente di esecuzione del Common Intermediate Language(CIL), il linguaggio intermedio in cui i compilatori della piattaforma .NET traducono i linguaggi ad alto livello come C#, oltre ad una serie di compilatori per ottenere il CIL. Trattandosi di una macchina virtuale, la cosiddetta Mono CLR è stata sviluppata per le varie piattaforme. Di nostro interesse sono quella per i sistemi operativi basati su Linux e BSD come Android, iOS e OS X e quella per i sistemi basati su Windows.

Il primo obiettivo di Xamarin è stato quello di creare librerie .NET che fungano da wrapper tra le API native di Android, iOS e Mac e il framework .NET, rendendo tali funzionalità raggiungibili dagli sviluppatori tramite l'utilizzo di un unico linguaggio (C#). Queste librerie sono:

- Xamarin.Android;
- Xamarin.iOS;
- Xamarin.Mac.

Significa che al momento le piattaforme supportate da Xamarin sono Android, iOS e Windows Phone. Tuttavia, l'ampliamento del supporto in seguito ad un interesse futuro può essere effettuato in maniera più o meno rapida aggiungendo esclusivamente una nuova libreria Xamarin.X visto che esistono già versioni della Mono CLR per le altre piattaforme presenti sul mercato.

2.2 C# come unico linguaggio

Il C# è un linguaggio di programmazione orientato agli oggetti sviluppato da Microsoft specificatamente per la programmazione nel framework .NET. I codici sorgente scritti in C# sono normalmente compilati secondo i criteri JIT(Just In Time): la trasformazione in codice macchina (ovvero eseguito direttamente dalla CPU) avviene su richiesta solo all'atto di caricamento ed esecuzione del programma. In prima istanza il codice sorgente viene convertito dal framework in un codice intermedio detto CIL e solo all'esecuzione del programma il CLR specifico per il sistema operativo utilizzato converte il CIL in codice macchina, man mano che viene eseguito. Xamarin sceglie C# come unico linguaggio permettendo agli sviluppatori di ottenere applicazioni native senza conoscere altri linguaggi come Objective C o Java e di rimanere in un unico ambiente di sviluppo. Per ottenere questo risultato le librerie sopracitate utilizzano meccanismi di binding e di linking di librerie native che corrispondono ad un mapping più o meno articolato a seconda della complessità di ogni funzionalità. Il valore aggiunto che si ottiene dal binding nativo è la possibilità di utilizzare nelle proprie applicazioni librerie scritte in altri linguaggi, cosa molto interessante quando si ha la necessità di utilizzare una libreria disponibile esclusivamente in un linguaggio specifico.

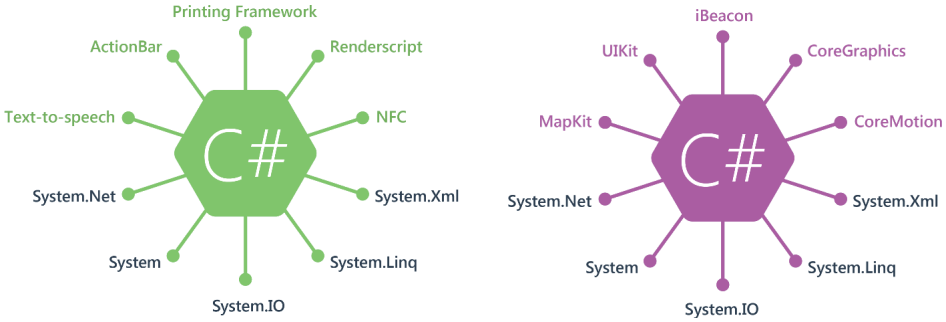


Figura 2.1: C#, alcune delle librerie raggiungibili[13]

2.2.1 Code sharing

Xamarin, tramite l'utilizzo delle librerie citate, permette di costruire una base comune alle varie piattaforme chiamata Shared C# Backend. Tale parte, a seconda dell'applicazione da realizzare, comprende una certa percentuale (che può arrivare anche al 100%) della logica di business e dell'accesso all'hardware. Sopra la base sarà poi realizzata un'interfaccia grafica per ogni piattaforma utilizzando i controlli nativi.

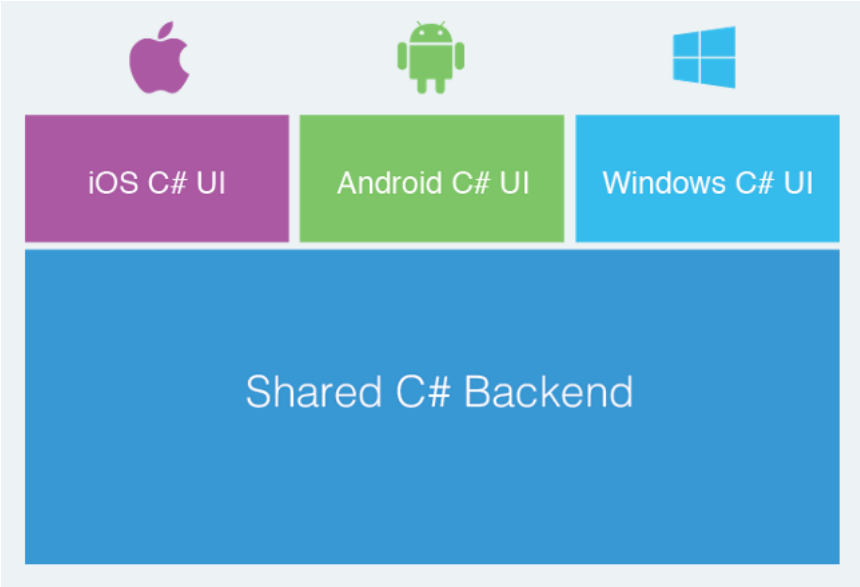


Figura 2.2: Approccio classico di Xamarin[6]

Trattandosi della parte più sostanziosa di una applicazione, Xamarin afferma che in media si hanno il 75% di righe di codice in comune[6].

2.2.2 Cross-compilazione

Il metodo con il quale si ottiene un'applicazione nativa da un codice C# varia a seconda della piattaforma:

- Windows Phone: Il codice C# viene compilato in IL(Intermediate Language). Per essere eseguito non vi è la necessità di alcun tool di Xamarin poiché è già presente nella piattaforma l'ambiente di esecuzione CLR .NET;
- Android (Xamarin.Android): Il codice C# viene compilato in IL e impacchettato con MonoCLR e JIT'ing. Quando l'applicazione viene eseguita, un compilatore JIT(Just in time compiler) analizza il bytecode identificando i punti nel quale è in grado di tradurre tale codice in istruzioni native.

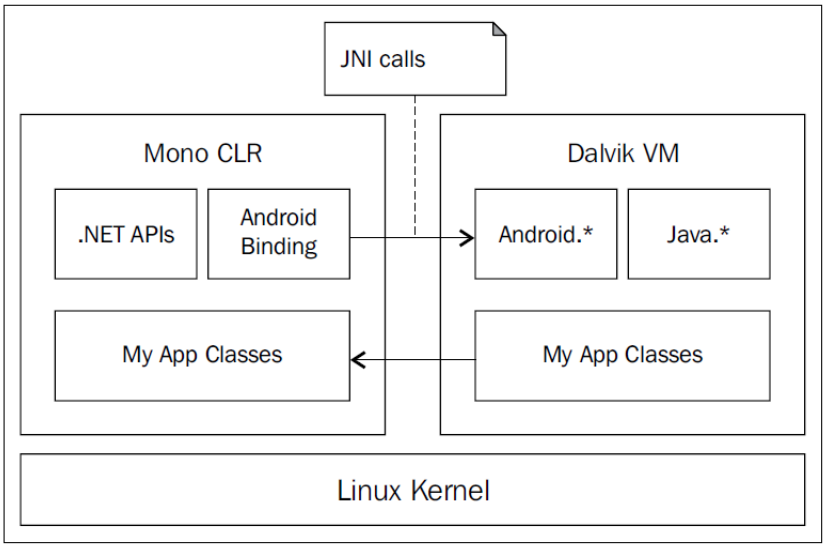


Figura 2.3: Macchine virtuali in Android[14]

Per l'esecuzione di applicazioni Xamarin.Android, Xamarin prevede l'utilizzo sia della MonoCLR che della DalvikVM. Queste due macchine virtuali vengono fatte comunicare tramite un framework chiamato JNI. Grazie a questa interazione, facendo uso di alcune tecniche di binding, Xamarin rende possibile l'esecuzione di codice java nativo usufruendo del linguaggio C#;

- iOS (Xamarin.iOS): A causa di alcune restrizioni di casa Apple, la generazione dinamica di codice tramite un compilatore JIT(Just in time

compiler) è proibita. Il codice C# viene compilato in IL e poi, dove è possibile, ricompilato in codice macchina (ARM assembly language) tramite un compilatore AOT(Ahead of time). Questo procedimento non elimina la necessità della macchina virtuale che viene comunque inclusa. L'uso della compilazione C# limita alcuni aspetti propri del linguaggio.

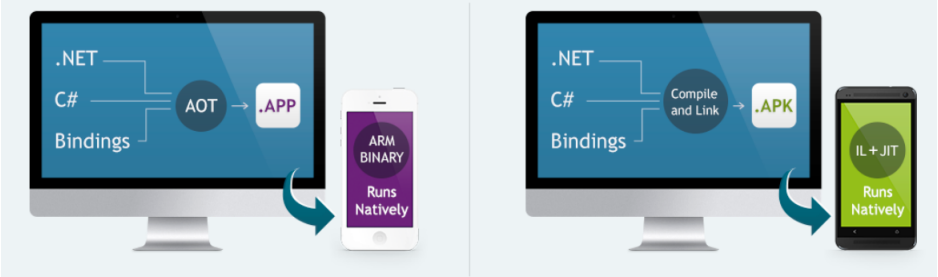


Figura 2.4: Generazione archivi binari[13]

2.3 Il nuovo approccio, Xamarin.Forms

Negli ultimi tempi Xamarin ha introdotto un nuovo framework per la creazione di applicazioni mobile chiamato Xamarin.Forms, che ha come obiettivo la costruzione di un sistema che permetta la condivisione di codice tra piattaforme a tutti i livelli, compreso quello riguardante le interfacce grafiche.

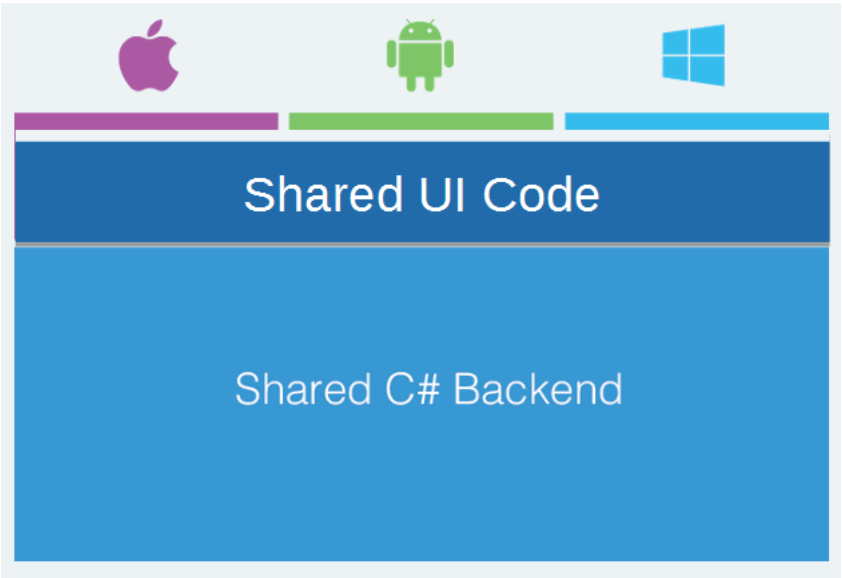


Figura 2.5: Nuovo approccio in Xamarin[15]

Il raggiungimento del 100% del codice condiviso è sicuramente un traguardo ambizioso, ma, a conferma che Xamarin è sulla giusta strada, vi è l'interesse che sta suscitando sul pubblico facendo crescere rapidamente il suo prodotto. Per rendere possibile tutto ciò, Xamarin.Forms fornisce un livello di astrazione che comprende tutti componenti che caratterizzano l'interfaccia grafica di un'applicazione, come: pagine, layout e controlli. Lo sviluppatore utilizzerà esclusivamente tali astrazioni per costruire l'interfaccia grafica della propria applicazione e, solo in fase di esecuzione, Xamarin.Forms si occuperà di fare il render andando a selezionare i corrispettivi elementi platform-specific necessari, ottenendo un look and feel nativo, ad esempio: il controllo *Entry* diventerà una *UITextView* in iOS, una *EditText* in Android e una *TextBox* su Windows Phone.

L'approccio utilizzato da Xamarin.Forms per la condivisione di codice è chiamato PCL (Portable Class Library) project. Tale approccio fa in modo che lo sviluppo dell'applicazione venga diviso in più progetti distinti, facenti parte però di un'unica soluzione. Sarà presente un progetto per ogni specifica piattaforma supportata, più uno che si trova ad un livello superiore rispetto agli altri definito *portatile*. Il nome lascia intendere che quest'ultimo racchiuda tutti quegli elementi comuni dello sviluppo (risorse, modelli, viste, ...) che, solo in fase di compilazione, vengono iniettati nei rispettivi progetti platform-specific che fanno uso delle librerie Xamarin.Android e Xamarin.iOS.

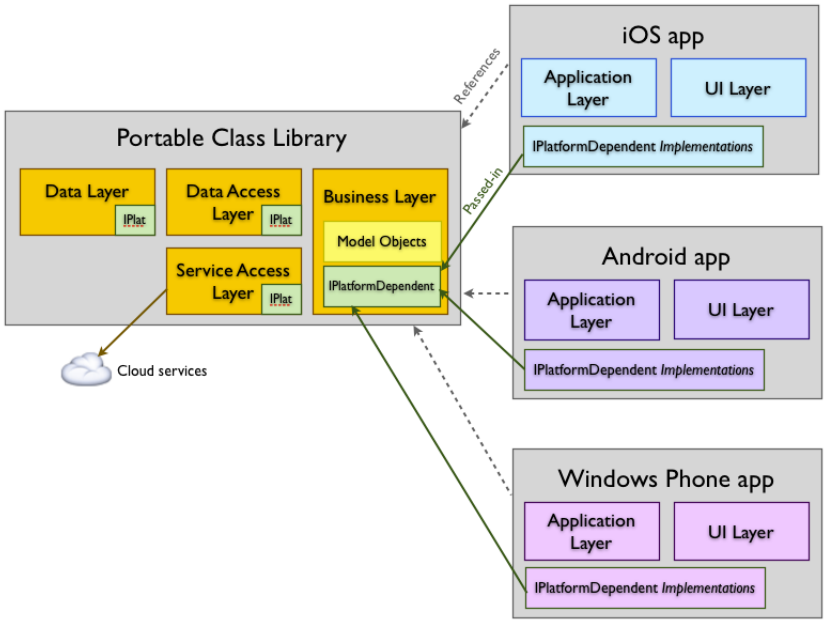


Figura 2.6: Progetto PCL[16]

Per poter effettuare questa operazione di inject, un progetto portable deve fornire una classe **App** rappresentazione di un'applicazione Xamarin.Forms. Questo genere di applicazione consiste in un susseguirsi di *pagine*, non ci sarà più la necessità di un diretto riferimento alle activity come in Xamarin.Android o alle *view* come in Xamarin.iOS, questo arriverà esclusivamente al momento di inject quando, nella *MainActivity* del progetto Android o nell'*AppDelegate* del progetto iOS o nella *MainPage* del progetto WindowsPhone, verrà eseguito il caricamento del progetto *portable* tramite la funzione *LoadApplication* di Xamarin.Forms. Per mantenere inalterata la gestione del ciclo di vita dell'applicazione nelle varie piattaforme, vengono forniti i metodi cross-platform: *OnStart*,*OnSleep* e *OnResume* raggiungibili dalla classe App. Sempre da qui viene definita quella che sarà la pagina principale dell'applicazione e il tipo di navigazione selezionabile tra quattro, come visibile in Figura 2.7.

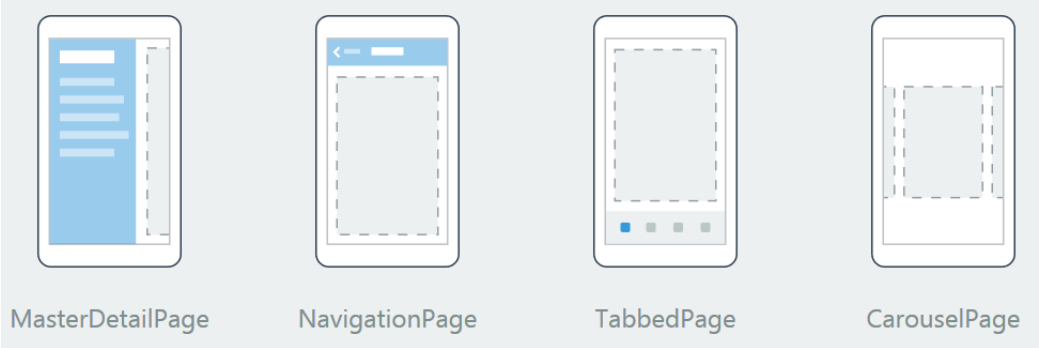


Figura 2.7: Tipologie di pagine e navigazione[17]

2.4 Strumenti per lo sviluppo

La suite di strumenti messi a disposizione da Xamarin è disponibile in licenza tramite una iscrizione annuale che prevede tre tipi di soluzioni: Indie, Busines e Enterprise. Il pacchetto Indie, al contrario degli altri, è ad uso individuale e non comprende l'assistenza tramite e-mail, ma consente di utilizzare tutti quei servizi che permettono la realizzazione e la pubblicazione di applicazioni senza alcun tipo di limitazione. Xamarin offre la possibilità di sviluppare le proprie applicazioni tramite un proprio ambiente di sviluppo chiamato *Xamarin Studio* o direttamente su *Visual Studio*, tramite l'aggiunta di un add-in che provvede all'installazione di Xamarin.iOS e Xamarin.Android. Quest'ultima è davvero ottima se si possiedono già conoscenze di tale ambiente poiché si andranno ad utilizzare gli stessi strumenti permettendo di essere produttivi già dal giorno uno. L'intero sviluppo di una applicazione Android può essere effettuato utilizzando Visual Studio, sia su macchina fisica che virtuale; per le

applicazioni iOS la fase di compilazione necessita della presenza di una macchina fisica Mac con installata la versione aggiornata di Xcode e dell' SDK iOS. Per gli utenti Visual Studio in ambiente Windows, Xamarin fornisce un servizio chiamato build server per poter compilare il proprio codice su una macchina Mac che si trova in rete per poi visualizzare il risultato sul PC. L'add-in per Visual Studio comprende strumenti che permettono di disegnare le interfacce grafiche delle proprie app, fornendo una toolbox contenente tutti i componenti grafici nativi.

Xamarin offre inoltre un servizio online dal nome *Xamarin Test Cloud* per effettuare dei test per la propria applicazione su migliaia di diversi dispositivi, mostrando per ognuno il risultato visivo e elencando i problemi verificatesi. Il cloud aiuta inoltre lo sviluppatore nel tener traccia di tutti i test effettuati e dei vari avanzamenti come dimensioni delle versioni e impatto sulla memoria dei dispositivi.

Capitolo 3

Caso di studio

In questo capitolo viene presentato il caso di studio, andando a definire il problema e il perché può essere considerato per valutare l'approccio e il framework selezionato.

3.1 Strumento per il supporto delle operazioni di soccorso

Come banco di prova per testare le potenzialità del framework per lo sviluppo di applicazioni multi-platform scelto, si andrà a realizzare un prototipo di applicazione Xamarin che corrisponderà ad un porting di un'applicazione platform-specific Android esistente. L'azienda che aveva commissionato il lavoro ha, negli ultimi tempi, sviluppato un interesse nel poter fornire i propri servizi ad utilizzatori di sistemi operativi mobile diversi da Android, occorre dunque trovare la soluzione ottima per poter permettere tutto ciò. Tale opportunità è perfetta per validare o meno una soluzione multi-platform poiché, essendo già stata effettuata una progettazione per una specifica piattaforma, nella fase di ri-progettazione sarà facile individuare gli aspetti che dovranno essere riconsiderati e che possono, ad esempio, portare la perdita di alcune funzionalità. Al termine della realizzazione del prototipo, sarà inoltre semplice paragonare le due diverse soluzioni.

3.1.1 Descrizione dello strumento

Lo strumento informatico fornito dall'azienda è pensato per i professionisti dell'emergenza a supporto delle operazioni di soccorso. L'obiettivo è quello di sostituire l'attuale approccio che vede l'utilizzo di materiale cartaceo da parte degli operatori di campo, sia per quanto riguarda i protocolli e le linee guida da

seguire, sia per la memorizzazione delle informazioni raccolte durante le operazioni di soccorso, come l'identità del paziente e i dati necessari a valutare il suo stato di salute. I benefici dell'introduzione di apparecchiature elettroniche per il problema del primo soccorso sono sicuramente importanti, in particolare il sistema fornisce i seguenti:

- Aiuto nel prendere le decisioni riducendo la possibilità di errori e di rischio clinico da parte dei soccorritori, guidando l'utilizzatore attraverso protocolli sempre aggiornati (manovre da eseguire, parametri da verificare, ecc. . .);
- Aumento della capacità di raccolta delle informazioni tramite l'utilizzo di sensori presenti nei dispositivi (fotocamera, gps, lettori di codici o tag, ecc. . .);
- Interazione automatica con i dispositivi per la raccolta dei dati clinici come: pressione arteriosa, SpO2, frequenza cardiaca ed elettrocardiogramma;
- Interazione real-time con la centrale operativa. I dati raccolti su campo possono essere facilmente trasferiti in centrale tramite Internet, in modo tale da verificare l'operato dei soccorritori, rapporto impossibile da ottenere tramite l'utilizzo di supporti cartacei;
- Aiuto nella predisposizione dell'evacuazione dei pazienti.

Per poter fornire tale servizio il sistema prevede due parti: una ad uso degli operatori di campo che consiste in un'applicazione per dispositivi mobili, una ad uso degli operatori in centrale che consiste in un'applicazione server-side. Il nostro interesse risiede nella prima. Verrà individuato un sottosistema abbastanza corposo della applicazione client, in modo tale da poter valutare la riuscita di un porting complessivo, obiettivo primario di questo elaborato.

3.2 Individuazione del sub-set di funzionalità da implementare

Come sub-set di funzionalità da implementare nel prototipo è stata selezionata la parte riguardante l'operazione di triage di primo livello. Per triage si intende il metodo che ha come scopo la valutazione delle priorità assistenziali di un paziente e quindi l'assegnazione di un codice colore, mentre per primo livello si intende che l'operazione viene svolta sul luogo di "crash" cioè dove l'incidente è avvenuto. Esistono diversi protocolli di triage di primo livello, il

sistema ne supporta più di uno ma, visto che l'implementazione tra questi è del tutto indifferente, si è deciso di supportarne nel prototipo solamente uno denominato protocollo S.T.A.R.T.(Simple triage and rapid treatment). Il triage di primo livello S.T.A.R.T. fornisce ai soccorritori un semplice approccio passo-passo, al fine di valutare e trattare in breve tempo un grande numero di pazienti con vari gradi di urgenza. Tali passi sono facilmente rappresentabili tramite un semplice diagramma di flusso:

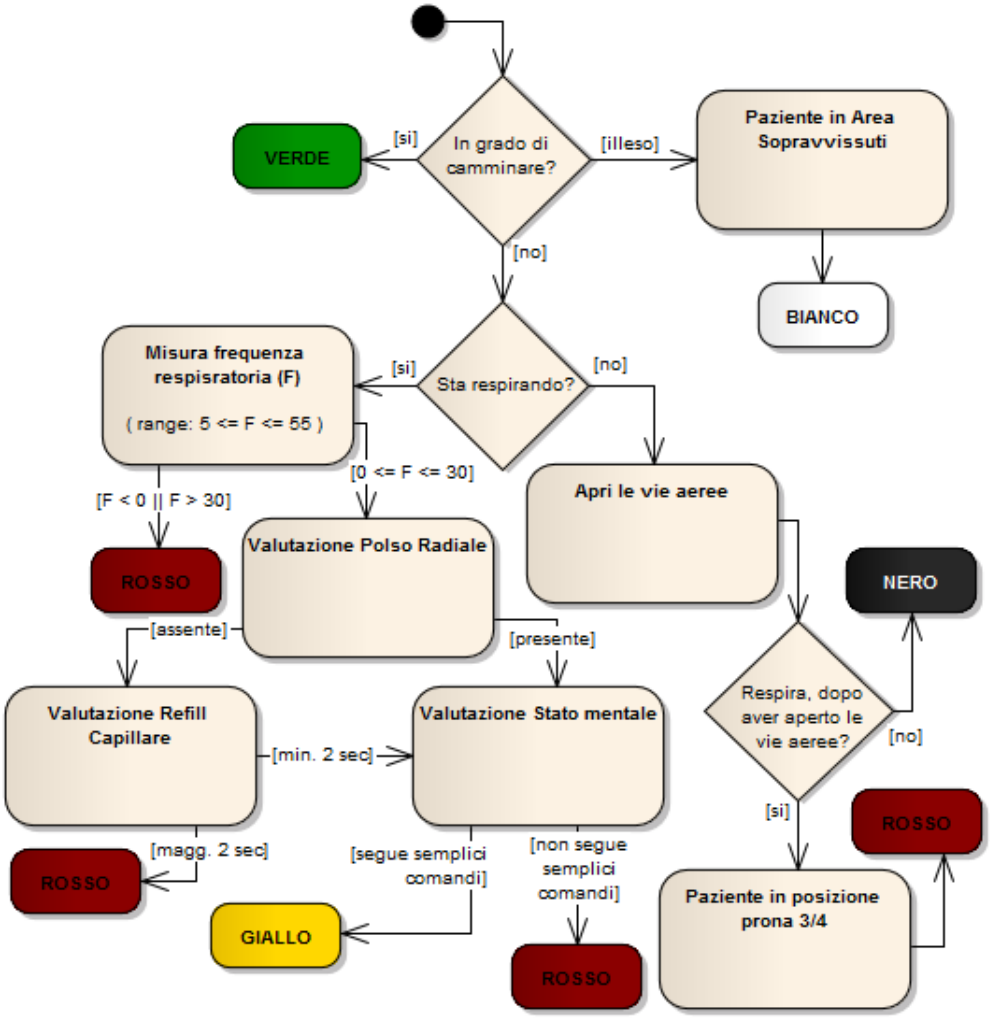


Figura 3.1: Diagramma di flusso dell'algoritmo S.T.A.R.T.

3.2.1 Requisiti dell'applicazione

Il prototipo dovrà offrire le seguenti funzionalità sui dispositivi mobili:

- autenticazione dell'operatore attraverso l'inserimento del proprio username e password;
- cattura di immagini atte a documentare lo stato di salute del paziente tramite fotocamera;
- ottenimento della posizione geografica dell'operatore al momento della compilazione del report;
- supporto decisionale all'attuazione delle operazioni di triage per la corretta esecuzione delle procedure di soccorso;
- consultazione dell'archivio storico delle operazioni eseguite.

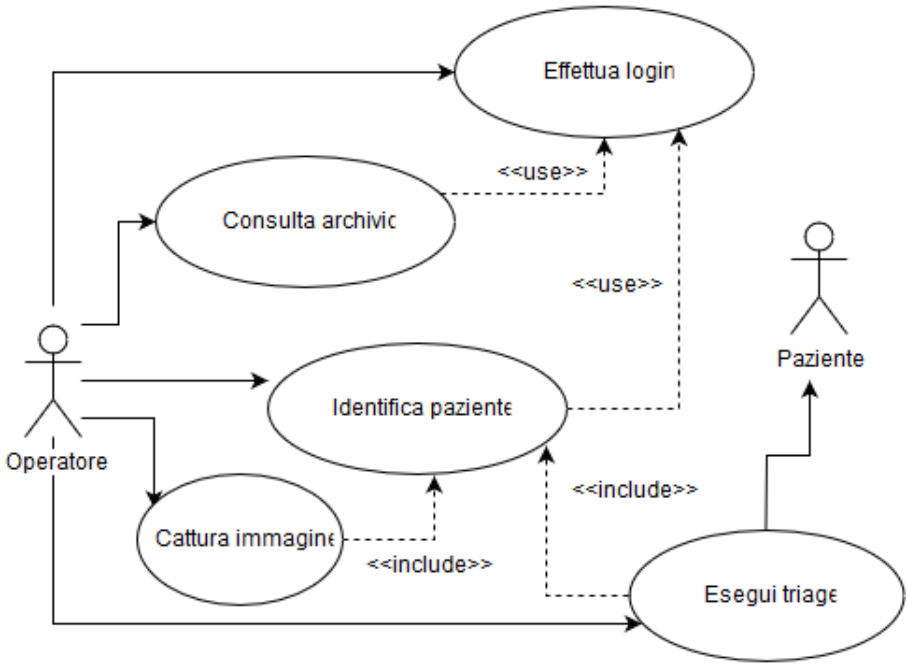


Figura 3.2: Diagramma casi d'uso

Visto l'ambito in cui tale software deve operare, le qualità fondamentali da considerare sono:

- correttezza e effidabilità: la salute del paziente assistito può dipendere dal corretto funzionamento dell'applicazione;
- efficienza: l'uso delle risorse deve essere efficiente in modo tale da non rallentare le operazioni di soccorso;

- facilità d'uso: lo scenario di utilizzo prevede situazioni critiche dovute alle condizioni di intervento, l'utilizzo da parte dell'operatore deve essere il più semplice possibile (tasti grandi, ecc...) per non essere di ostacolo nella corsa contro il tempo.

3.2.2 Parametri di valutazione della soluzione multi-platform

La valutazione finale della soluzione multi-platform sarà risultato di verifiche effettuate in due fasi del progetto. La prima, in fase di realizzazione, offrirà una panoramica delle difficoltà incontrate nell'ottenere le varie funzionalità del sistema rispetto all'approccio platfrom-specific, elogiando le parti che vedono la completa condivisione di codice tra le diverse piattaforme; la seconda, in fase di test, corrisponderà al confronto di prestazioni e di look & feel con l'applicazione platform-specific Android esistente.

Capitolo 4

Progettazione ed implementazione del caso di studio

In questo capitolo viene mostrato il processo di progettazione, sviluppo e test dell'applicazione evidenziando le differenze rispetto all'utilizzo di un approccio nativo per poi trarre le conclusioni del lavoro svolto.

4.1 Progettazione

Dall'analisi dei requisiti effettuata emerge la necessità di sviluppare un prototipo di applicazione mobile, con lo scopo di valutare se la tecnologia multi-platform, individuata come la più adatta, abbia raggiunto un grado di maturazione tale da poter essere utilizzata per progetti di grandi dimensioni. Il prototipo dovrà replicare, nella maniera più fedele possibile, le funzionalità messa a disposizione dall'applicazione Android esistente per il sub-set individuato come più significativo, cioè quello di triage. Trattandosi di uno sviluppo che vede l'utilizzo di una tecnologia multi-platform, è interessante spingere al massimo sul concetto di condivisione del codice quindi, anche dove sarà possibile ottenere risultati migliori utilizzando codice platform-specific, verrà data sempre priorità ad una soluzione platform-indipendente funzionante, facendo le dovute precisazioni. Il framework che si presta meglio a questo obiettivo, e per questo utilizzato, è **Xamarin.Forms**.
Per comprendere al meglio come può avvenire il passaggio da una progettazione nativa in Android ad una multi-platform, prima di descrivere l'architettura del prototipo verrà presentata una breve descrizione dell'architettura del sistema preesistente.

4.1.1 Architettura del sistema esistente per le parti interessate

Trattandosi di un'applicazione software che inizialmente era destinata ad essere distribuita esclusivamente su piattaforma Android, il progetto risulta essere diviso in componenti ottenuti estendendo i cosiddetti *building blocks* offerti dall'infrastruttura Android stessa. Si tratta di un'architettura orientata alla definizione di gran parte dei componenti del sistema sotto forma di *activity*, che vanno a definire interfaccia utente e parte del comportamento, soprattutto grazie alla possibilità di interscambio di *fragments* come è ben visibile nel blocco di maggior rilievo della Figura 4.1 rappresentante l'architettura del sistema.

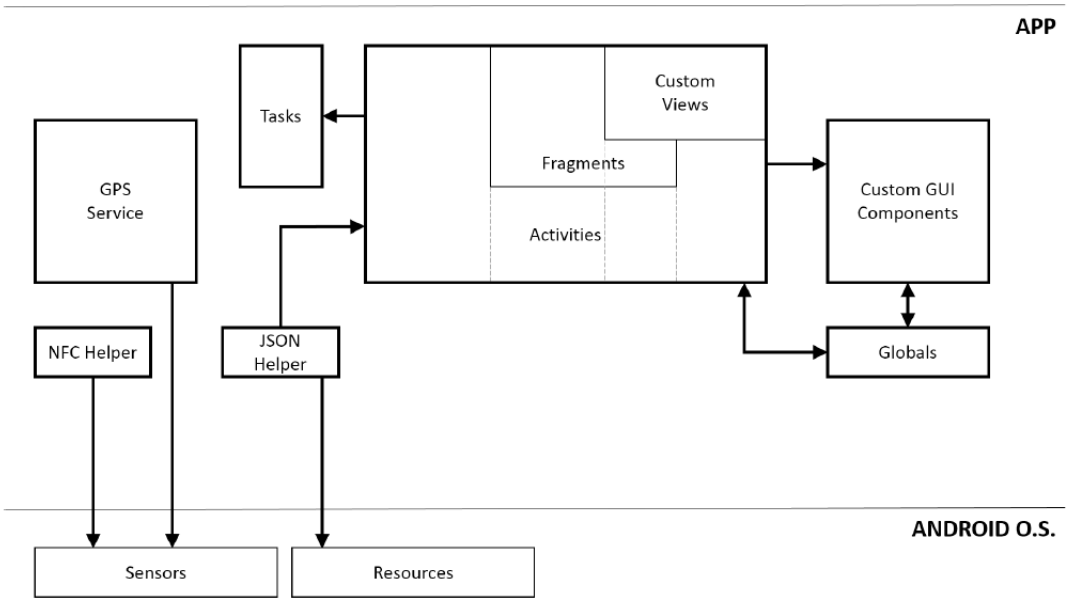


Figura 4.1: Architettura sistema preesistente

L'architettura dell'applicazione risulta più chiara se vista in più parti: una principale comprendente tutte quelle activity generiche del sistema, e poi una per ogni protocollo supportato. La prima, indipendente dai vari protocolli, fungerà da punto di accesso alle altre denominate moduli. Nei paragrafi successivi sarà presentato il modello architetturale di navigazione del sistema software delle varie parti.

Architettura di accesso e configurazione Nella parte principale, descritta in Figura 4.2, è presente lo *start point* dell'applicazione. La *MainActivity*

presenta la possibilità di effettuare il login ed accedere alle funzionalità del sistema raggiungibili tramite la *ConfigurationActivity*.
Compito della MainActivity è anche quello di avviare i servizi di localizzazione.

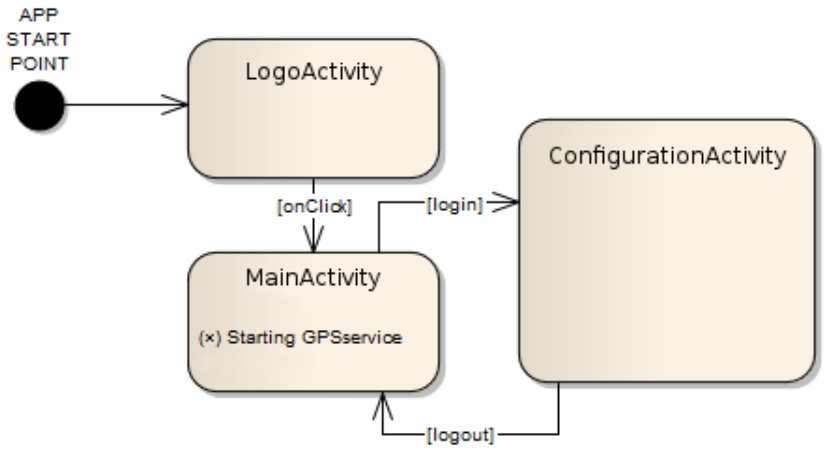


Figura 4.2: Navigazione parte accesso e configurazione

Architettura modulo di triage La parte di triage rappresenta uno dei moduli raggiungibili tramite l'activity *ConfigurationActivity*. All'avvio del protocollo S.T.A.R.T. l'utente può procedere all'identificazione del paziente tramite l'activity *PatientActivity*. Qui deve essere possibile associare al paziente un codice univoco ed aggiungere eventuali altre informazioni, come immagine rappresentativa dello stato di salute del paziente o codice identificativo QR.

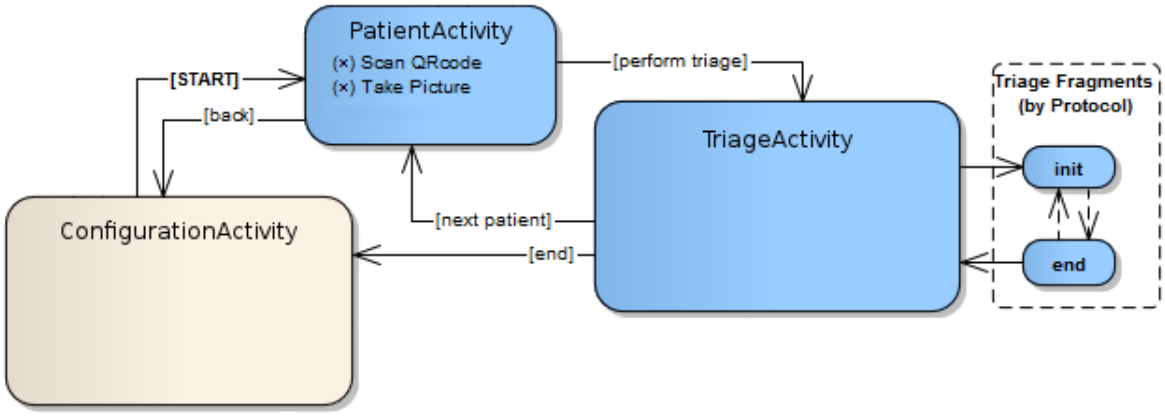


Figura 4.3: Navigazione modulo di triage

Una volta terminata la fase di identificazione, è possibile iniziare la fase di triage passando all'activity *TriageActivity* che, tramite un interscambio di

fragments, riproduce l'algoritmo corrispondente al diagramma di flusso visto in Figura 3.1.

Architettura modulo di storage La parte di storage raccoglie tutte le informazioni raccolte relative ai pazienti. Il modello di navigazione è presentato nella Figura 4.7. Partendo dalla *StorageActivity* è possibile selezionare un paziente che è già stato sottoposto ad almeno un processo di triage per visualizzare i risultati ottenuti nella *SelectionActivity*. Da quest'ultima è possibile eseguire altre procedure medico-sanitarie attivando il modulo di triage.

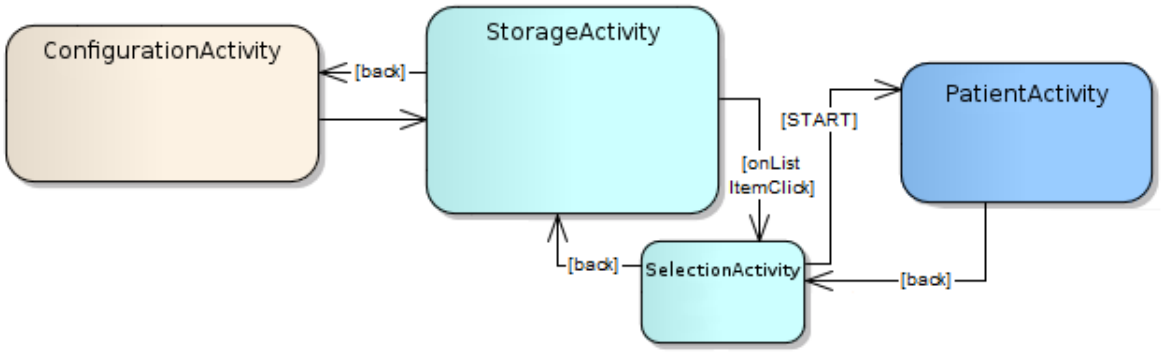


Figura 4.4: Navigazione modulo storage

4.1.2 Architettura del nuovo sistema

Come descritto nel Capitolo 2, le applicazioni create con Xamarin.Forms presentano un'architettura orientata alla definizione di gran parte dei componenti del sistema sotto forma di *pagine*, che vanno a definire interfaccia utente e parte del comportamento. Nella ri-progettazione del sistema è quasi sempre possibile far corrispondere una *Android Activity* con una *Xamarin.Forms Page*, occorre invece trovare un'alternativa all'utilizzo degli *Android Fragments*. Xamarin.Forms non fornisce infatti elementi con la stessa funzione, o simile, di quest'ultimi.
Vengono ora presentati i modelli di navigazione del nuovo sistema per le parti precedentemente individuate.

Architettura di accesso e configurazione Il modello architetturale della parte principale è quasi del tutto analogo a quello del sistema preesistente. L'unica differenza riguarda lo *start point* dell'applicazione che corrisponde con la *LoginPage*. Da qui è possibile effettuare l'accesso alle funzionalità dell'applicazione raggiungibili tramite la *ConfigurationPage*.

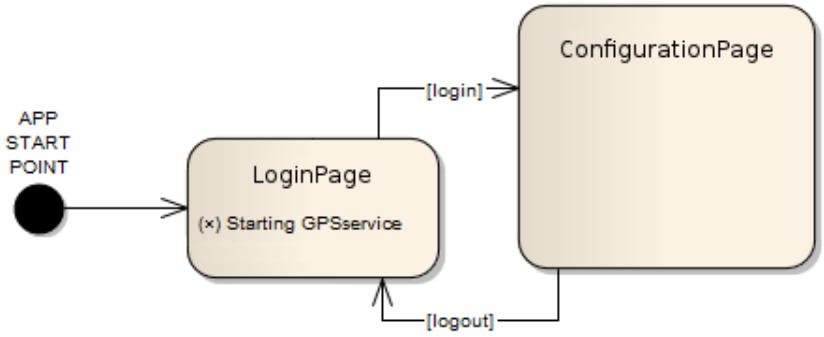


Figura 4.5: Navigazione parte accesso e configurazione

Architettura modulo di triage Il modello architetturale di questo modulo risulta essere del tutto analogo a quello del sistema preesistente, tranne per la parte riguardante all'effettiva esecuzione dell'algoritmo di triage. L'incapacità di sostituire il contenuto di una pagina in modo programmatico, porta al dover definire una pagina per ogni step dell'algoritmo, quindi all'impossibilità di racchiudere l'intera logica di funzionamento in un unico luogo come avveniva per la *TriageActivity*. Tale activity, rimanendo attiva per tutto il processo di triage, poteva contenere sia la logica di presentazione degli step del triage, sia le informazioni raccolte. Nella nuova soluzione, visibile in Figura 4.10, si fa uso di una pagina sprovvista di interfaccia grafica denominata *TriageRoutePage*, che ha il compito di richiamare sistematicamente la pagina rappresentante il prossimo step da eseguire.

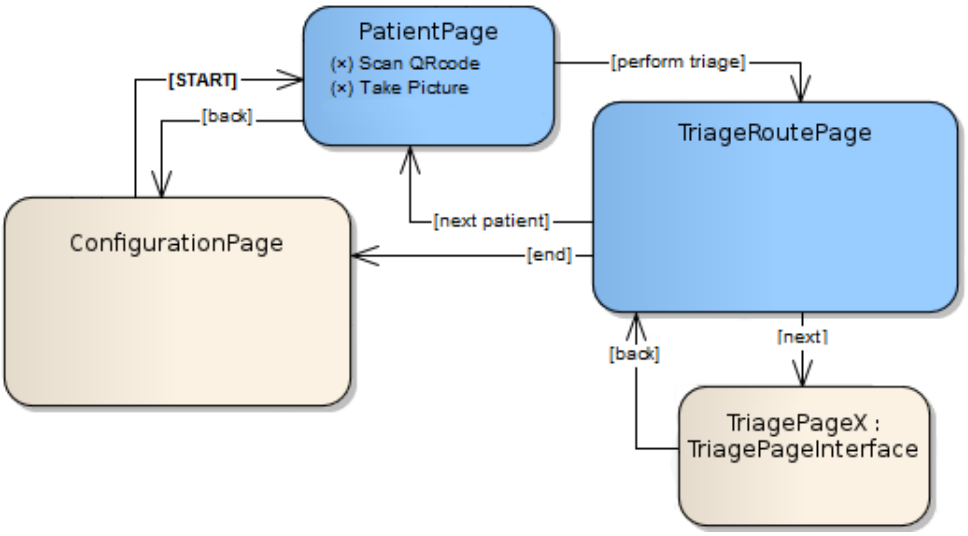


Figura 4.6: Navigazione modulo di triage

Sfruttando il ritorno alla *TriageRoutePage* al termine di uno step, viene mantenuta l'esperienza di navigazione del sistema preesistente.
Per evitare la proliferazione di pagine, dovuta alla moltitudine di domande e azioni a cui l'utente può essere sottoposto, è stata definita una pagina per ogni tipologia di step individuata nella sezione 3.2. Queste pagine si alterneranno cambiando il loro contenuto all'occorrenza per formare quello che è il processo di triage.

Architettura modulo di storage La parte riguardante il modulo di storage può essere ottenuta effettuando semplicemente il passaggio *Activity to Page* visto nei paragrafi precedenti.

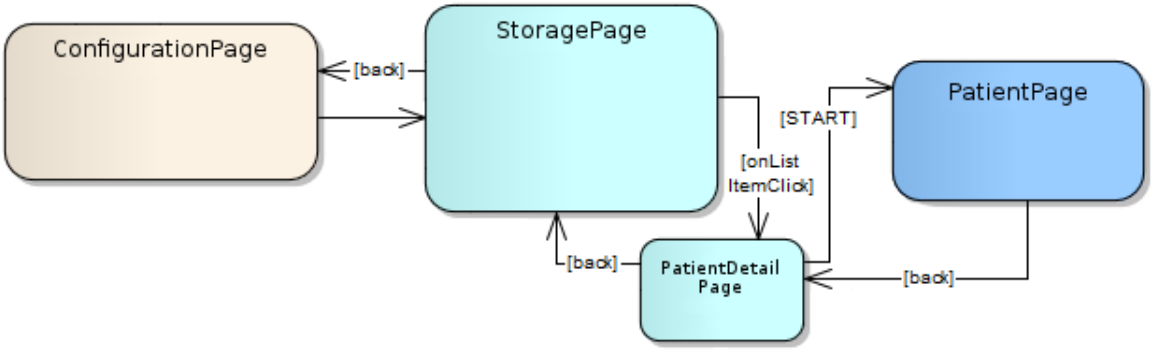


Figura 4.7: Navigazione modulo di storage

4.1.3 Modello concettuale

Dall'analisi sulla realtà da rappresentare risulta possibile definire il dominio del problema tramite tre entità principali: il paziente, la sessione di triage e l'operatore. Il paziente, definito esclusivamente da un codice, può essere sottoposto ad uno o più sessioni di triage. Una sessione di triage è eseguita da un solo operatore (soccorritore) ed è caratterizzata da una data e ora di inizio, una data e ora di fine, una posizione geografica e il risultato del triage, cioè il colore rappresentante la gravità dello stato del paziente. L'operatore è caratterizzato da un username e una password per poter effettuare il login al sistema.

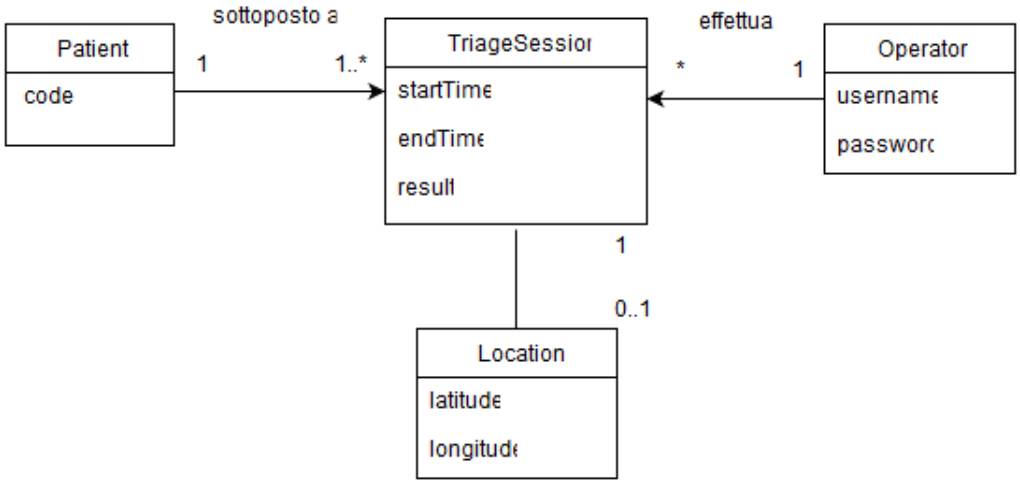


Figura 4.8: Diagramma delle classi

4.2 Implementazione

L'implementazione dell'applicazione è stata effettuata in ambiente Windows 8.1 utilizzando l'IDE Visual Studio 2015, integrato con lo strumento per lo sviluppo di applicazioni multi-platform Xamarin.Forms. La presenza dell'add-in permette la creazione di nuovi progetti *portable* definiti nella sezione 2.3, andando a specificare nome del progetto che in questo caso è **TriageApp**. Come piattaforme da supportare sono state selezionate:

- Android API Level 23;
- iOS 9;
- Windows Phone 8.1.

Per poter effettuare quest'ultima operazione è stato necessario installare nel sistema le relative SDK. Quello che è stato generato è una "soluzione" contenente quattro progetti come mostrato in Figura 4.9

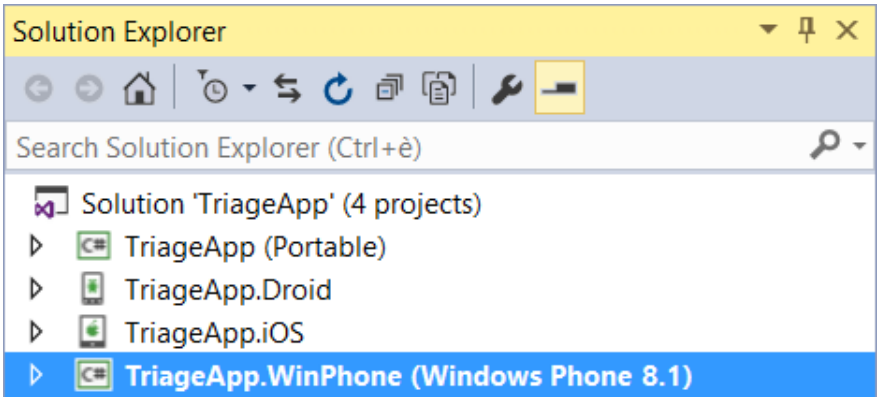


Figura 4.9: Progetti in VisualStudio

Da questo momento si utilizzerà il nome **TriageApp.Droid** per riferirsi al progetto Android, **TriageApp.iOS** per iOS e **TriageApp.WinPhone** per Windows Phone, mentre per il progetto comune esclusivamente **TriageApp**. Per ognuno di questi progetti ho deciso di effettuare l'aggiornamento all'ultima versione stabile *2.0.1.6505* di Xamarin.Forms tramite il gestore di pacchetti NuGet disponibile in Visual Studio. Sempre tramite quest'ultimo sono state aggiunte ai progetti le librerie open-source **XLabs** alla versione *2.0.5782*, per permettere l'interazione con i sensori del dispositivo.

4.2.1 Struttura generale del progetto TriageApp

Per organizzare al meglio le parti del progetto in comune, è stato scelto di suddividere il suo contenuto per separare concettualmente i vari moduli individuati in fase di progettazione ottenendo come risultato i seguenti *namespace*:

- TriageApp: contiene esclusivamente la classe App.cs rappresentante la radice di un'applicazione cross-platform;
- TriageApp.Model: contiene le classi che rappresentano i modelli individuati in fase di progettazione;
- TriageApp.ModelView: contiene le classi che corrispondono al Model-View del pattern MVVM;
- TriageApp.Pages: contiene tutte le pagine dell'applicazione suddivise per scopo;
- TriageApp.Pages.TriageSTARTPages: contiene tutte le pagine che intervengono nel corso del processo di triage;

- TriageApp.Pages.StoragePages: contiene tutte le pagine necessarie a visualizzare i dati raccolti dai vari processi di triage;
- TriageApp.Utils: contiene le utility dell'applicazione comprese quelle necessarie a simulare degli elementi non richiesti al prototipo ma comunque necessari al funzionamento di alcune delle sue parti.

Come modello di navigazione tra pagine è stato ritenuto opportuno adottare esclusivamente quello a stack, che vede la definizione di una *RootPage* che sta alla base della navigazione.

Figura 4.10: Navigazione a stack

Navigare verso una nuova pagina corrisponderà ad inserirla in cima alla stack, mentre navigare indietro corrisponderà alla sua rimozione. Per definire questo modello è sufficiente impostare come MainPage dell'applicazione una nuova *NavigationPage* alla quale passiamo la RootPage che, dalla progettazione dell'architettura, è risultata essere la *LoginPage*.

```
using TriageApp.Pages;

namespace TriageApp
{
    public class App : Application
    {
        public App()
        {
            // The root application
            MainPage = new NavigationPage(new LoginPage());
        }
    }
}
```

App.cs

In questo modo sono stati attivati i meccanismi di push e pop in tutte le pagine del progetto, raggiungibili attraverso l'utilizzo dei metodi *PushAsync* e *PopAsync*.

4.2.2 Da XML a XAML

Per riprodurre le view della applicazione nativa esistente è stato effettuato, senza alcuna difficoltà, un procedimento manuale di sostituzione di tag XML di Android con tag XAML di Xamarin.Forms analoghi. Segue un confronto tra i layout grafici per ottenere la form della login.

```
<LinearLayout
   android:orientation="vertical">
    <TextView
      android:text="Username"/>
    <EditText
        android:id="@+id/edit_username"/>
    <TextView
        android:id="@+id/label_pass"
        android:text="Password"/>
    <EditText
        android:inputType="textPassword"
        android:id="@+id/edit_password"/>
    <Button
        android:id="@+id/button_login"
        android:text="Login" />
</LinearLayout>
```

MainActivity.XML rappresentante la login del sistema preesistente

```
<StackLayout>
   <Label
     Text="Username" />
    <Entry
     x:Name="Username_entry"/>
    <Label
     Text="Password" />
    <Entry
     x:Name="Password_entry"
     IsPassword="true" />
    <Button
     Text="login"
```

```
    Clicked="OnLoginPressed" />
</StackLayout>
```

LoginPage.XAML rappresentante la nuova login cross-platform

Segue una tabella di conversione degli elementi grafici di Android in elementi grafici Xamarin.Forms.

Tipo	**Android**	**cross-platform**
Layout	LinearLayout	StackLayout
	AbsoluteLayout	AbsoluteLayout
	RelativeLayout	RelativeLayout
	GridLayout	GridLayout
Controlli	Button	Button
	EditText	Entry
	TextView	Label
	Spinner	Spinner
	CheckBox	-
	RadioButton	-
	ImageView	Image
	MapView	Map
	DatePicker	DatePicker
	TimePicker	TimePicker
	ListView	ListView
	Slider	-

Tabella traduzione da XML a XAML

Per ovviare alla mancanza di alcuni controlli fondamentali non ancora forniti da Xamarin.Forms, sono state utilizzate delle implementazioni open-source presenti nelle librerie XLabs.

4.2.3 Implementazione parte accesso e configurazione

La parte di accesso e configurazione è composta da due pagine: la *LoginPage* e la *ConfigurationPage*. Come visto nella progettazione, la *ConfigurationPage* corrisponde ad una semplice schermata di accesso ai vari moduli. Più interessante è la *LoginPage*, da qui viene attivato il servizio di geo-localizzazione ed effettuato l'accesso introducendo la questione dell'esecuzione in parallelo. Per eseguire il login senza bloccare la view, è stato definito un task asincrono a cui è stato demandato il compito di verifica delle credenziali. Tale operazione, essendo in ambiente .NET, è sufficiente per garantire il parallelismo in Xamarin.

```
async void OnLoginPressed(object sender, EventArgs e)
{
    String username = Username_entry.Text;
    String password = Password_entry.Text;
    Task<bool> loginTask = LoginAsync(username, password);
    bool result = await loginTask;

    if (result) {
        Navigation.PushAsync(new ConfigurationPage());
    }else{
        DisplayAlert("Attenzione", "Username o password
            errati", "OK");
    }
}

public async Task<bool> LoginAsync(String username, String
    password)
{
    await Task.Delay(1000);
    if (SingletonFakeData.singleton.login(username,
        password)){
        return true;
    }
    return false;
}
```

LoginPage.xaml.cs

Nell'implementazione del servizio di geo-localizzazione, l'unica operazione platform-specific necessaria, è stata quella di ottenere il permesso per l'acquisizione della posizione attraverso le proprietà dei progetti *TriageApp.Droid*, *TriageApp.iOS*, *TriageApp.WinPhone*. Tramite un'istanza della classe *Geolocator* fornita dalla libreria XLabs è possibile conoscere la latitudine e la longitudine dell'operatore richiamando il metodo *GetPositionAsync*.

```
private async Task GetPosition()
{
    await
        Geolocator.GetPositionAsync(10000,
            _cancelSource.Token, true)
            .ContinueWith(t =>
            {
```

```
                    PositionStatus =
                        t.Result.Timestamp.ToString("G");
                    PositionLatitude = "La: " +
                        t.Result.Latitude.ToString("N4");
                    PositionLongitude = "Lo: " +
                        t.Result.Longitude.ToString("N4");
                }, _scheduler);
    }
```

Un problema di questa implementazione emerge quando l'applicazione viene posta in background. In questo caso l'aggiornamento della posizione non avviene. Non è ancora possibile un'implementazione cross-platform per la registrazione di servizi in backgorund, sono solo presenti delle utility che aiutano a definirli[18].

4.2.4 Implementazione modulo di triage

Il modulo di triage è composto dalla *PatientPage*, dalla *TriageRoutePage* e dalla serie di pagine rappresentanti le tipologie di step individuate nell'algoritmo S.T.A.R.T. che sono:

- l'*ActionPage*: rappresenta un'azione che l'operatore deve compiere, al termine di questa, l'unica possibilità è quella di procedere con il prossimo step;
- la *DoubleAnswerPage*: rappresenta una domanda con due possibili risposte;
- l'*InputPage*: rappresenta una azione che richiede l'inserimento di una misurazione;
- la *ResultPage*: rappresenta un risultato del processo di triage;
- la *TripleAnswerPage*: rappresenta una domanda con tre possibili risposte.

Per permetter la cattura di immagini quando si è nella *PatientPage*, non è necessario l'utilizzo di codice platform-specific. Tramite un'istanza della classe *MediaPicker* fornita dalla libreria XLabs, è possibile richiamare la fotocamera del dispositivo con il metodo asincrono *TakePhotoAsync*.

```
private async Task<MediaFile> TakePicture()
        {
            ImageSource = null;

            return await _mediaPicker.TakePhotoAsync(new
                CameraMediaStorageOptions { DefaultCamera =
                CameraDevice.Front, MaxPixelDimension = 400
                }).ContinueWith(t =>
            {
                    var mediaFile = t.Result;

                    ImageSource = ImageSource.FromStream(() =>
                        mediaFile.Source);

                    return mediaFile;
            }, _scheduler);
        }
```

Per implementare l'algoritmo di routing descritto nella progettazione, si fa uso del pattern creazionale signleton tramite la classe *SingletonTriageSTART*. All'avvio del percorso di triage tale classe viene istanziata e, tramite la lettura di un file JSON, vengono definiti quelli che sono i passi corrispondenti all'algoritmo S.T.A.R.T.. Il file JSON è formattato in modo tale da contenere una serie di oggetti, ognuna rappresentante un blocco del diagramma di flusso di Figura 3.1. L'applicazione inizia cercando tra i vari oggetti JSON quello che corrisponde al blocco iniziale. Per convenzione tale oggetto si chiama *inizio* e contiene un campo di nome *prossimo* il cui valore corrisponde al primo blocco decisionale da mostrare.

```
{
   "inizio": {
        "testo": "",
        "prossimo": "cammina",
        "tipo": 0
    },
    "cammina": {
      "testo": "Riesce a camminare?"
        "no": "respira",
        "si": "verde",
        "altrimenti": "illeso",
        "tipo": 6
    },"verde": {
```

```
        "testo": "non grave",
        "tipo": 1
    },"respira": {
        "testo": "Riesce a respirare?",
        "no": "apriVieAeree",
        "si": "calcolaRespiri",
        "tipo": 2,
    },
    ...
    ...
}
```

File JSON rappresentante l'algoritmo S.T.A.R.T.

La *TriageRoutePage* chiede all'istanza del *SingletonTriageSTART* qual è la prossima pagina da richiamare tramite il metodo *getNextTriageStep*. Ottenuta la risposta, a seconda del tipo del blocco da visualizzare, viene chiamata la nuova pagina passandogli la domanda o l'azione da mostrare e le varie possibili risposte.

```
private void PageCaller()
        {
            TriagePageLogic pageLogic =
                SingletonTriageSTART.singleton.getNextTriageStep();
            switch (pageLogic.type)
            {
                case 0:
                    break;
                case 1:
                    Navigation.PushAsync(new
                        ResultPage(pageLogic.text));
                    break;
                case 2:
                    Navigation.PushAsync(new
                        DoubleAnswerPage(pageLogic.text,
                        pageLogic.textBtn1, pageLogic.textBtn2));
                    break;
                case 3:
                    Navigation.PushAsync(new
                        ActionPage(pageLogic.text,
                        pageLogic.textBtn1));
                    break;
```

```
            case 4:
                Navigation.PushAsync(new InputPage(pageLogic.text,
                    pageLogic.textBtn1));
                break;
            case 5:
                break;
            case 6:
                Navigation.PushAsync(new
                    TripleAnswerPage(pageLogic.text,
                    pageLogic.textBtn1, pageLogic.textBtn2,
                    pageLogic.textBtn3));
                break;
        }
    }
```

TriageRoutePage.cs

Tutte le interazioni comportano delle azioni svolte nell'istanza del *SingletonTriageSTART*, al termine delle quali la pagina viene rimossa avendo un ritorno alla *TriageRoutePage* che richiede la prossima pagina da chiamare fino a che l'operatore non decida che il processo di triage sia concluso. Per permettere l'annullamento delle risposte date dall'operatore, l'istanza del *SingletonTriageSTART* contiene il percorso effettuato, comprensivo dei risultati ottenuti in modo da poter gestire il "torna indietro", contemplato in tutte le pagine del triage tramite la pressione del tasto "indietro" che corrisponde alla chiamata del metodo *getBackTriageStep*.
La scelta di utilizzare un documento esterno permette, in caso di variazione dell'algoritmo S.T.A.R.T., di modificare il comportamento del modulo di triage senza dover ricorrere a modifiche nel codice.

4.3 Test e debug

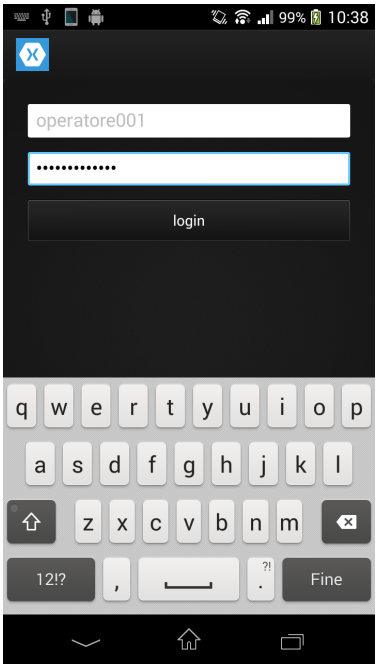
L'attività di test è stata eseguita, in maniera manuale, utilizzando device fisici o simulati in ambiente desktop e, in maniera automatica, tramite il servizio *Xamarin Test Cloud*. Per l'indisponibilità di dispositivi iOS e l'impossibilità di indicare a Visual Studio una macchina virtuale OSX per il deploy di applicazioni Xamarin.iOS, ha portato all'esclusione dai test la piattaforma iOS. I test manuali, a differenza di quelli su cloud, sono stati effettuati anche durante il processo di implementazione. I dispositivi utilizzati sono:

- Sony Xperia V, 1GB RAM, 4.3", Android 4.3;

- Motorola Moto G 3rd, 1GB RAM, 5.0", Android 5.1.1;
- simulatore Nexus 5, 1GB RAM, 4.95", Android 6.0;
- emulatore Windows Phone 8.1 U1, 512MB RAM, 4.0".

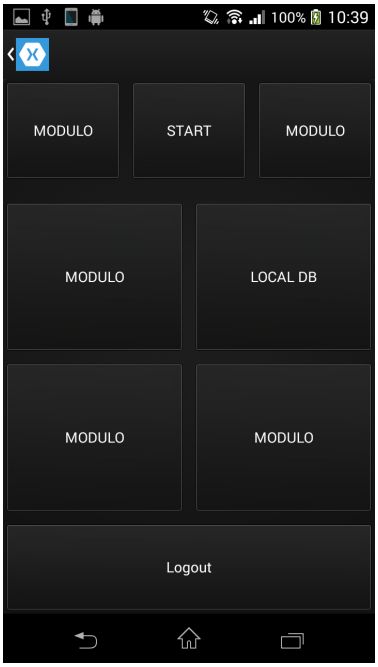
Per motivi che porterebbero al dover secretare la tesi, non è possibile riportare gli screenshot dell'applicazione Android preesistente. Il risultato ottenuto tramite il processo di replicazione dell'interfaccia grafica è ottimo su tutti i dispositivi considerati, per questa ragione l'esperienza di utilizzo viene considerata del tutto inalterata.

Seguono alcuni screenshot dell'applicazione in eseguzione su Sony Xperia V e sull'emulatore di Windows Phone.

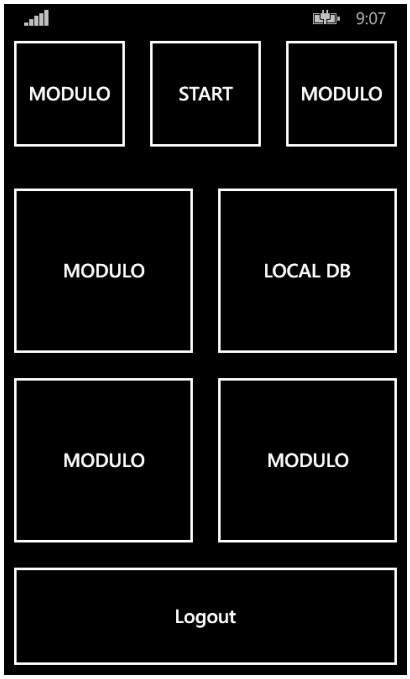


(a) LoginPage, TriageApp.Droid

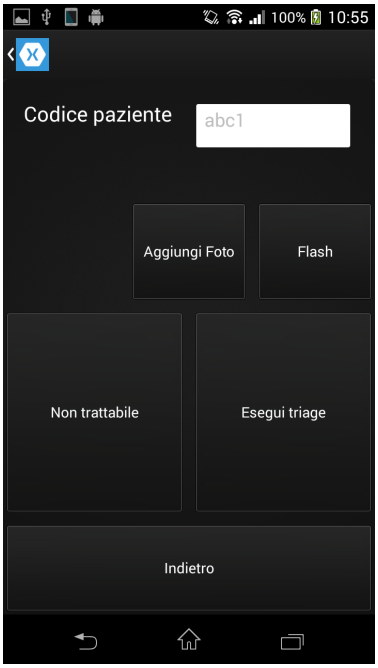
(b) LoginPage, TriageApp.WinPhone



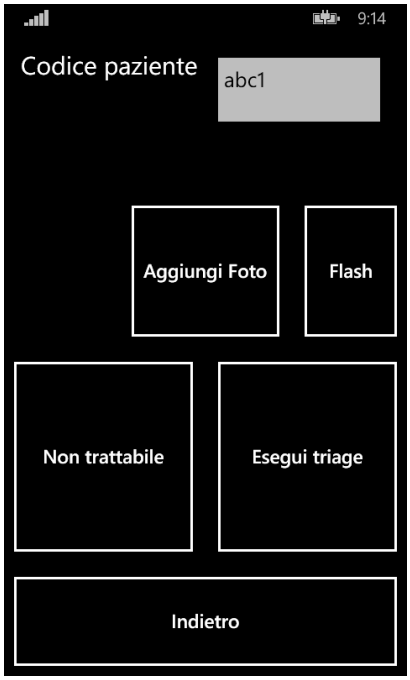
(a) ConfigurationPage, TriageApp.Droid



(b) ConfigurationPage, TriageApp.WinPhone

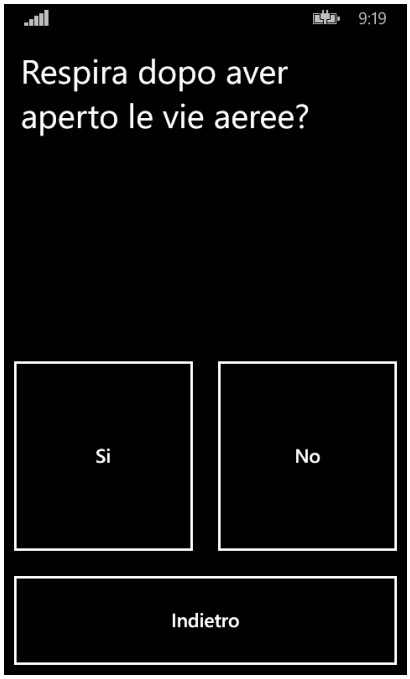


(a) PatientPage, TriageApp.Droid

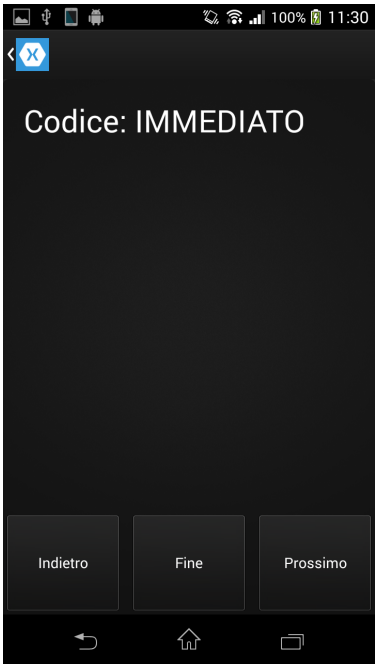


(b) PatientPage, TriageApp.WinPhone

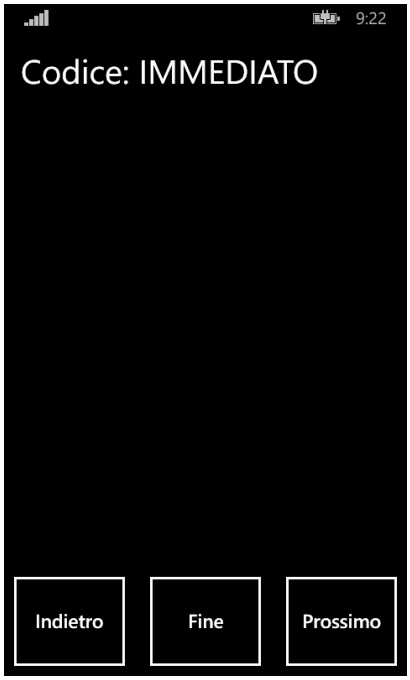
(a) DoubleAnswerPage, TriageApp.Droid



(b) DoubleAnswerPage, TriageApp.WinPhone



(a) ResultPage, TriageApp.Droid



(b) ResultPage, TriageApp.WinPhone

Dai test eseguiti manualmente non sono stati riscontrati problemi di alcun tipo. Sia l'esecuzione dell'applicazione *TriageApp.Droid* che quella dell'applicazione *TriageApp.WinPhone*, risulta fluida in tutto il suo corso compresa la parte di utilizzo dei sensori e la parte più macchinosa di routing delle pagine durante il processo di triage.
I test automatici eseguiti tramite il tool *Xamarin Test Cloud*, sono risultati molto utili per la risoluzione di alcuni bug a fine implementazione per quanto riguarda l'applicazione *TriageApp.Droid*. E' stato possibile ampliare notevolmente il set di dispositivi sulla quale ottenere dei feedback. In particolare ho selezionato novanta dispositivi (settantuno smartphone e diciannove tablet) individuati tra quelli maggiormente diffusi. Ho inoltre travato opportuno aggiungere anche i modelli corrispondenti ai device fisici di cui dispongo. L'applicazione, nel suo stato ultimo, risulta avere una corretta esecuzione su ottantuno dispositivi selezionati, mentre viene riscontrato almeno un problema nei rimanenti nove. I problemi individuati sono di due tipi: uno viene intercettato quando, una volta scattata la foto, si ritorna alla pagina *PatientPage*, l'altro è invece di tipo grafico è causato dalle ridotte dimensioni dello schermo (es. Samsung Galaxy NEO 3"). I dispositivi in cui il primo problema causa un crash dell'applicazione sono: *Amazon Kindle Fire HD, Amazon Kindle Fire, Samsung Galaxy Note 2, Samsung Google Nexus 10,Samsung Galaxy Win Duos*.

Marca	**Dispositivi con problemi**
Acer	0/2
Amazon	2/2
Asus	0/5
HTC	0/6
HUAWEI	0/1
LG	2/12
Motorola	0/6
Nokia	0/1
OnePlus	0/1
Samsung	4/40
Sony	1/14

Dispositivi con problemi, divisi per marca

Nonostante le ricerche effettuate sulle cause, non si è riusciti a trovare una soluzione e, non disponendo dei corrispettivi device fisici non è stato possibile verificare l'effettiva occorrenza del problema che comunque non risulta essere presente nei test effettuati manualmente.

4.4 Considerazioni finali e sviluppi futuri

L'obiettivo di realizzare il prototipo facendo uso limitato di codice platform-specific, è stato raggiunto a pieno. Il processo di produzione del software è risultato essere al 100% cross-platform, soprattutto grazie all'ambiente di sviluppo Visual Studio con le sue modifiche automatizzate ai file propri dei progetti *TriageApp.Droid*, *TriageApp.iOS* e *TriageApp.WinPhone*. Viene confermata la possibilità di realizzazione di applicazione non banali da parte di sviluppatori che possiedono conoscenze minime delle piattaforme target. Lo sviluppo del prototipo è stato relativamente semplice, non si sono evidenziate particolari difficoltà rispetto alla realizzazione platform-specific Android, se non per una forte mancanza che è l'indisponibilità di un tool grafico a supporto della produzione delle view che porta al dover ricompilare ed eseguire la soluzione, per poter apprezzare modifiche a volte minime. Il compromesso più grande che ho dovuto affrontare è stato quello di non poter apprezzare le peculiarità di ogni singola piattaforma a favore di una soluzione completamente cross-platform, dato che lo stile di programmazione tramite Xamarin.Forms si è adattato, secondo me, in maniera eccessiva a quello di Windows Phone, le cui applicazioni presentano una struttura più rigida rispetto a quelle degli altri sistemi operativi mobile. Il punto in cui questo si è fatto maggiormente sentire è stato durante lo sviluppo del modulo di triage, dove l'impossibilità di sostituire il contenuto di una pagina ha portato l'obbligo di dirigersi in altre costringendo la definizione di un'istanza di una classe condivisa da utilizzare per le comunicazioni. Il risultato ottenuto, se pure meno elegante, replica comunque in maniera ottima l'esperienza di utilizzo dell'app nativa Android che utilizza dei fragments. Molto positiva è stata l'esperienza avuta con la community di utilizzatori del framework a fronte di mancanze nella documentazione ufficiale che, a mio avviso, risulta essere rivolta troppo verso lo "stile tutorial" a dispetto della spiegazione dei funzionamenti interni dei vari componenti. Ho molto apprezzato il lavoro open-source svolto con le librerie XLabs, diventate di riferimento per la gestione dei sensori su applicazioni Xamarin.Forms anche se non comprendo il perché un framework, il cui utilizzo prevede una spesa cospicua per l'acquisto della licenza, non si preoccupi di fornire delle librerie proprie migliorando l'integrazione con i propri strumenti e documentazione. Ad indicare la dubbia posizione presa da Xamarin vi è la completa assenza di riferimenti alle librerie XLabs e all'interazione con i sensori tramite Xamarin.Forms nella documentazione ufficiale che può portare perplessità nello sviluppatore, che si trova a dover scegliere il framework per i propri scopi. Sono comunque soddisfatto del comportamento di Xamarin per la realizzazione del progetto, di fondamentale importanza è stata la presenza di un linguaggio compilato quale C# che ha permesso la semplice individuazione di errori che,

con i linguaggi interpretati utilizzati da framework alternativi, sono visibili solo runtime. Per lo stato in cui si trova oggi il sistema da riprodurre, viene confermata la possibilità di un passaggio completo utilizzando Xamarin.Forms dopo aver effettuato un approfondito studio su quelli che possono essere gli sviluppi futuri. Uno dei motivi che porterebbe all'esclusione di questa soluzione potrebbe essere, ad esempio, la volontà di ottenere delle interfacce grafiche personalizzate per ogni piattaforma a sfavore del code-sharing oppure, la forte necessità di utilizzo di funzionalità native. In questo caso Xamarin consiglia l'uso esclusivo di Xamarin.Android e XamariniOS anche per la definizione delle UI, ricordiamo la copertura totale e sempre aggiornata delle API native, ottenibile utilizzando in modo diretto, e non tramite Xamarin.Forms, di quest'ultimi framework. Un aspetto che mi piacerebbe approfondire in futuro, che per ragioni di tempo non viene affrontato in questo elaborato, è l'integrazione con il database locale delle varie piattaforme.

Conclusioni

Può essere considerato raggiunto il primo obiettivo, quello di costituire una base decisionale a supporto dello sviluppatore, che si accinge a selezionare l'approccio a lui più adatto.
Sono stati individuati pregi e difetti dei vari approcci, ma, soprattutto, sono state evidenziate quelle mancanze, dove presenti, che portano ad una esclusione immediata della soluzione multi-platform.
Per quanto riguarda la convalidazione del framework Xamarin, strumento di sviluppo multi-platform individuato come più interessante tra quelli presenti oggigiorno, si sono ricevuti feedback molto positivi, rafforzati dalla completa riuscita della realizzazione del prototipo corrispondente al caso di studio trattato nel capitolo 3. Ulteriori informazioni a riguardo sono disponibili nella sezione 4.4 *Considerazioni finali e sviluppi futuri*.
La mia considerazione personale è: nella situazione attuale, il raggiungimento del 100% di condivisione del codice tra piattaforme, pur essendo possibile per la realizzazione di applicazioni non banali, porta ad una semplificazione eccessiva dei costrutti cross-platform, i quali vengono adattati alle capacità della piattaforma meno avanzata tra quelle supportate, andando a limitare le capacità delle altre. Tuttavia l'interesse presente per le tecnologie di questo genere risulta essere molto forte, ciò mi porta a credere che ne conseguirà un rapido processo di consolidazione che comprenderà anche quest'ultimo tema.

Bibliografia

[1] http://www.forbes.com/2009/02/19/innovation-internet-health-entrepreneurs-technology_wharton.html

[2] http://www.gartner.com/newsroom/id/2939217

[3] https://www.internetretailer.com/2014/12/10/mobile-apps-dominate-time-consumers-spend-online

[4] http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/

[5] https://www.internetretailer.com/2014/03/10/mobile-commerce-will-be-nearly-half-e-commerce-2018

[6] https://xamarin.com/platform

[7] http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[8] http://www.franzrusso.it/condividere-comunicare/mobile-2012-android-primo-sistema-operativo-in-italia/

[9] http://appsonmob.com/hybrid-app-webview-performance-ios-android/

[10] https://cordova.apache.org/plugins/

[11] https://www.dartlang.org/

[12] http://www.typescriptlang.org/

[13] http://www.slideshare.net/DemianFlavius/c-everywhere-building-crossplatform-apps-with-xamarin-and-mvvmcross

[14] http://xamarintutorials.com/Tutorial/TutorialDetail/2-Xamarin-Android-Architecture

[15] http://www.slideshare.net/Xamarin/introduction-to-xamarinforms

[16] https://developer.xamarin.com/guides/cross-platform/application_fundamentals/pcl/introduction_to_portable_class_libraries/

[17] https://xamarin.com/forms

[18] http://arteksoftware.com/backgrounding-with-xamarin-forms/

[19] Charles Petzold, *Creating Mobile Apps with Xamarin.Forms*, Microsoft Press, 2015.

[20] Mark Reynolds, *Xamarin Essentials*, Packt Publishing, 2014.