ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA M

TESI DI LAUREA

in

Mobile Systems M

SVILUPPO DI UN'APPLICAZIONE MOBILE CROSS PLATFORM MEDIANTE TECNOLOGIA XAMARIN

CANDIDATO:

Federico Gava

RELATORE:

Prof. Paolo Bellavista

CORRELATORE:

Prof. Luca Foschini

Ing. Giuseppe Martuscelli

Ing. Vincenzo Carnazzo

Anno Accademico 2018/2019

Sessione II

Indice

Introduzione

La rivoluzione digitale a cui stiamo assistendo in questi ultimi decenni interessa un numero sempre maggiore di ambiti, i quali mutano e si adattano in base alle tecnologie disponibili [1].

Anche il settore della didattica è stato coinvolto da questo cambiamento. I nuovi metodi di insegnamento, infatti, prevedono l'utilizzo crescente di strumenti digitali a supporto dei metodi di insegnamento tradizionali, come ad esempio i computer, le lavagne interattive multimediali (LIM) e software creati appositamente a sostegno di docenti ed alunni nelle attività scolastiche.

Tra le aziende italiane software house che si occupano della creazione di programmi indirizzati al mondo della didattica, la cooperativa Anastasis si occupa di studenti con Disturbi Specifici dell'Apprendimento (DSA) e di disabilità, con prodotti e servizi online per l'autonomia nello studio, la riabilitazione, la scuola e l'apprendimento. Anastasis nasce a Bologna nel 1985 dall'idea di Giovanni Zanichelli, già fondatore di A.S.P.H.I. (all'epoca Associazione per lo Sviluppo Professionale degli Handicappati in campo Informatico) per la formazione professionale in campo informatico di persone non vedenti e audiolese. Ancora oggi, obiettivo primario dell'azienda è intervenire con la propria esperienza e professionalità nei settori che prevedono l'uso delle nuove tecnologie e dei sistemi cosiddetti di punta a favore dei disabili e delle persone svantaggiate [2].

Tra i software prodotti dalla società bolognese a supporto di bambini e ragazzi nel loro percorso scolastico troviamo SuperMappe.

SuperMappe è un programma per creare mappe multimediali attraverso l'utilizzo di forme, colori e immagini. Il software è uno strumento utile ad una didattica inclusiva e collaborativa, e può essere utilizzato per aumentare l'autonomia e potenziare il metodo di studio dei bambini con DSA e difficoltà di apprendimento e per chiunque desideri un supporto visivo allo studio [3].

Anastasis ha nel tempo realizzato diverse versioni dell'applicazione SuperMappe: alcune rivolte ai sistemi desktop, altre ai sistemi mobili.

In ambito informatico un problema che ricorre con sempre maggior frequenza è la diffusione di dispositivi informatici molto diversi fra loro che richiedono quindi versioni differenti dello stesso applicativo. Ogni versione dovrebbe essere quindi sviluppata

sfruttando appieno le caratteristiche di ogni piattaforma portando però il codice delle diverse versioni a divergere, rendendo difficoltosa la sua manutenzione e l'inserimento di nuove funzionalità. Di conseguenza si verifica un incremento delle ore di sviluppo da parte dei programmatori con una conseguente crescita dei costi di realizzazione che le aziende produttrici di software devono sostenere.

In alternativa, l'applicazione potrebbe essere realizzata in maniera tale da essere compatibile con qualsiasi dispositivo, non riuscendo però a sfruttare completamente le caratteristiche e i punti di forza di ogni piattaforma.

In aggiunta, ogni versione viene scritta utilizzando l'ambiente di sviluppo (IDE) tipico di della piattaforma destinataria dell'applicativo che si sta sviluppando. Per iOS lo sviluppo viene realizzato solitamente con Xcode utilizzando il sistema operativo MacOS, per Android viene utilizzato Android Studio e per lo sviluppo di applicazioni Windows viene impiegato Visual Studio.

Per ovviare a queste problematiche, nel 2011 è stata fondata negli Stati Uniti l'azienda Xamarin con l'obiettivo di permettere agli sviluppatori di realizzare applicazioni native Android, iOS e Windows condividendo il codice su diverse piattaforme usando lo stesso ambiente di sviluppo [4].

Scopo di questa tesi è realizzare l'applicazione SuperMappe utilizzando la tecnologia *Xamarin* in modo tale che il software prodotto possa essere eseguito indistintamente sui dispositivi mobili, sia con sistema operativo iOS che Android.

La tesi è strutturata come segue. Nel primo capitolo viene descritto il funzionamento di SuperMappe; il secondo capitolo è invece dedicato alla descrizione della tecnologia Xamarin: segue nel terzo capitolo il progetto e la sua implementazione.

Il quarto e ultimo capitolo riassume infine le conclusioni e i possibili sviluppi futuri.

1. SuperMappe

SuperMappe è un software sviluppato dalla società Anastasis dedicato principalmente ad alunni ed insegnanti con il quale è possibile creare, personalizzare e gestire mappe concettuali in modo semplice [3].

Questo software è stato sviluppato per ambiente desktop Windows con il nome di *SuperMappe EVO* e successivamente ne è stata realizzata una versione *web-based* chiamata *SuperMappeX* riportata in Figura 1.

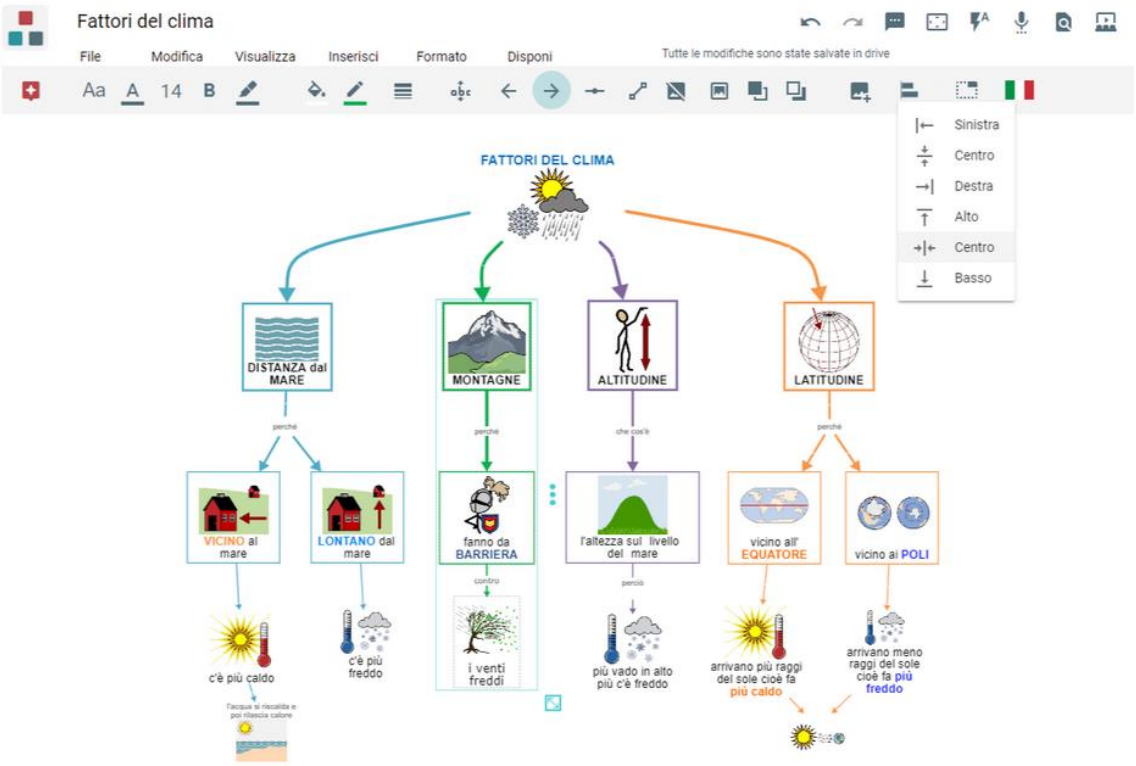

Figura 1: software SuperMappeX

In passato sono state inoltre realizzate delle versioni dedicate al sistema operativo Android impiegando diverse tecnologie e linguaggi come Ionic, HTML5 e Typescript.

SuperMappeX si presta ad un utilizzo in classe, attraverso l'utilizzo di videoproiettori o LIM, nell'ambito di una didattica collaborativa per presentare argomenti e per favorire confronti significativi, e per un utilizzo autonomo nello studio individuale quotidiano degli studenti [5].

Come per le mappe concettuali tradizionali, SuperMappe prevede la creazione di nodi collegati tra loro tramite archi.
I nodi rappresentano un concetto e possono essere costituiti da del testo semplice o da delle immagini. Ad essi è possibile aggiungere del testo di approfondimento e link a pagine web visualizzabili su richiesta interagendo con il nodo stesso.
I nodi sono messi in relazione tra loro attraverso gli archi rappresentati da frecce come evidenziato nella Figura 2.

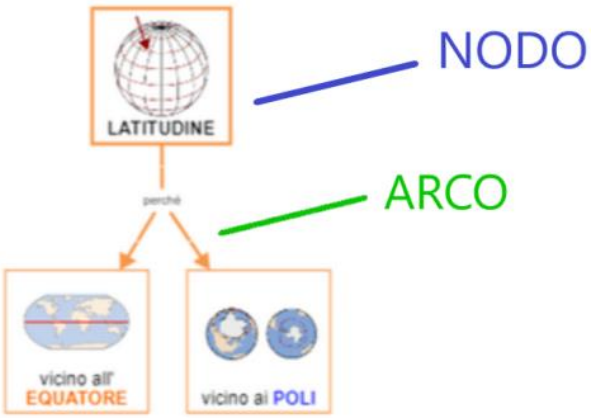

Figura 2: nodi e archi all'interno dell'applicazione SuperMappeX

SuperMappe consente la personalizzazione dell'aspetto grafico dei nodi e degli archi e permette di salvare le mappe create nello spazio cloud Google Drive.
Inoltre, è presente la sintesi vocale per permettere di inserire del testo tramite il riconoscimento vocale (*speech to text*) oppure di ascoltare i testi presenti nella mappa (*text to speech*).
La modalità presentazione permette poi di studiare i contenuti della mappa introducendo gradualmente i contenuti, modalità che può tornare utile anche nelle presentazioni in classe.

2. Tecnologia Xamarin

La tecnologia Xamarin, prodotta dall'omonima azienda acquisita da Microsoft nel 2016, permette di creare applicazioni native per le diverse piattaforme a partire da una singola *codebase* condivisa.

In questo modo non è più necessario utilizzare ad esempio il linguaggio di programmazione Objective-C per scrivere l'applicazione destinata a dispositivi iOS, Java per la versione dedicata a dispositivi Android e C# per Windows, ma è possibile scrivere l'applicazione interamente in un linguaggio comune come C# o in alternativa F#.

Di conseguenza, con l'utilizzo di questa tecnologia è possibile sviluppare un'applicazione per dispositivi mobili compatibile con le principali piattaforme esistenti: Android, iOS e UWP (Universal Windows Platform). In aggiunta è presente il supporto per tvOS e watchOS (sistemi operativi basati su iOS dedicati rispettivamente ai dispositivi Apple TV ed Apple Watch) e per Tizen utilizzato sui dispositivi Samsung tra cui televisori, dispositivi indossabili, dispositivi mobili e IoT.

Attualmente è inoltre presente il supporto in anteprima per GTK#, macOS e WPF (Windows Presentation Foundation).

Lo sviluppo di un'applicazione con tecnologia Xamarin può avvenire utilizzando due approcci differenti chiamati *Xamarin Native* e *Xamarin.Forms*.

Con il termine Xamarin Native, si indica lo sviluppo di codice nativo specifico per una singola piattaforma.

Ad esempio, Xamarin.Android costituisce la parte di progetto per lo sviluppo specifico di un'applicazione destinata ai dispositivi con sistema operativo Android in cui vi è possibile richiamare qualsiasi metodo esposto dall'omonimo SDK. Allo stesso modo, Xamarin.iOS fornisce agli sviluppatori l'accesso all'SDK completo di iOS.

Xamarin.Android e Xamarin.iOS sono entrambi progettati sulla base di Mono, una implementazione open source di Microsoft .NET Framework basata sugli standard aperti .NET ECMA.

In Android il compilatore di Xamarin consente di compilare nel linguaggio intermedio (IL), che viene quindi compilato Just-in-Time (JIT) in assembly nativo all'avvio dell'applicazione.

In iOS invece, il compilatore Ahead Of Time (AOT) di Xamarin compila le applicazioni Xamarin.iOS direttamente nel codice assembly ARM nativo.

In entrambi i casi, il risultato è indistinguibile da quello prodotto con gli ambienti di sviluppo predefiniti delle due piattaforme e quindi i pacchetti APK e IPA potranno essere distribuiti negli store e nei dispositivi esattamente allo stesso modo [6].

Xamarin.Forms invece, lavora al di sopra di Xamarin.Android e Xamarin.iOS e permette di realizzare interfacce utente multipiattaforma in C# e XAML (Extensible Application Markup Language).

Tramite questo approccio, viene offerto allo sviluppatore un sottoinsieme delle API comune alle diverse piattaforme richiesto per scrivere la gran parte di un'app in una codebase unificata [7].

Ad esempio, nel caso in cui si voglia visualizzare un interruttore all'interno dell'interfaccia grafica dell'applicazione, è possibile inserire nel codice condiviso l'elemento *Switch*. La classe Switch è implementata nella libreria Xamarin.Forms.Core e quest'ultima richiamerà i componenti nativi di ciascuna piattaforma a seconda dei casi. La classe Switch è mappata nella classe *UISwitch* nel caso di iOS, *Switch* per la piattaforma Android e nel *ToggleSwitch* per UWP.

In Figura 3 viene riportata l'interfaccia grafica di una semplice applicazione composta da un pulsante, da un interruttore e da un dispositivo di scorrimento nelle diverse piattaforme.

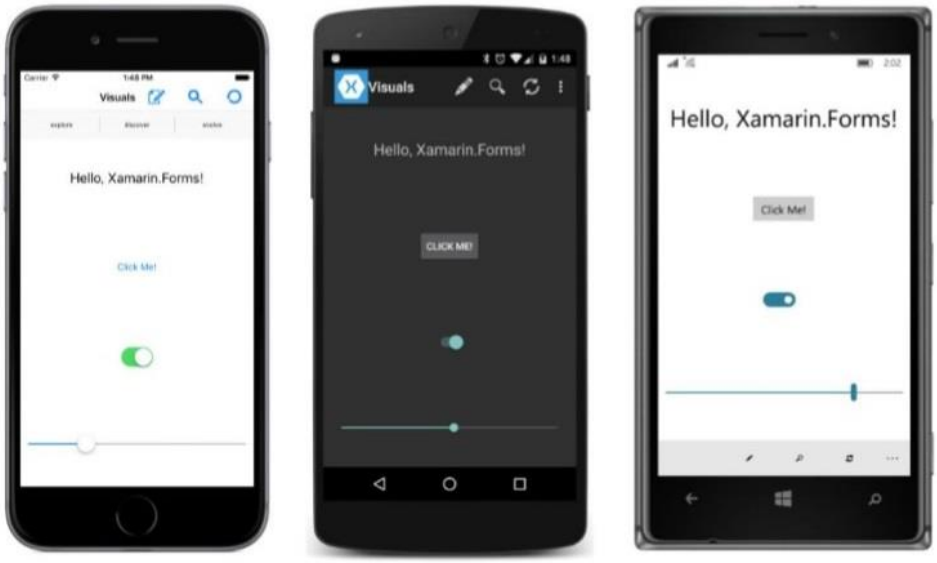

Figura 3: aspetto grafico di un'applicazione nelle piattaforme iOS, Android e UWP

Sebbene Xamarin.Forms fornisca i principali elementi grafici per la realizzazione di un'applicazione, non comprende tutti i componenti presenti in iOS, Android e UWP.
I due approcci Xamarin Native e Xamarin.Forms però non sono mutuamente esclusivi. Un'applicazione è realizzabile in gran parte utilizzando Xamarin.Forms ed è possibile intervenire solo in specifici punti con Xamarin Native dove richiesto.
In questo modo si possono ottenere i vantaggi di ciascun approccio, ovvero la possibilità di condividere la maggior parte del codice tra i diversi progetti e la possibilità di accedere all'SDK completo per realizzare limitate parti dell'applicazione.
Nella Figura 4 vengono mostrati i due approcci Xamarin Native e Xamarin.Forms a confronto. A sinistra si può notare come, sebbene si sia riusciti a condividere il codice di business logic, le interfacce grafiche sono implementate separatamente per ciascuna piattaforma. A destra invece è stato utilizzato Xamarin.Forms ed è stato possibile condividere sia il codice relativo alla business logic sia quello dedicato all'interfaccia grafica. Rimane visibile una sottile striscia colorata per ciascuna piattaforma ad indicare come in alcuni contesti sia comunque necessario intervenire separatamente per ciascuna piattaforma.

Figura 4: confronto tra Xamarin Native (a sinistra) e Xamarin.Forms (a destra)

Un altro elemento importante in Xamarin quasi indispensabile per lo sviluppo di un'applicazione è la libreria *Xamarin.Essentials.* Essa offre agli sviluppatori API multipiattaforma per le applicazioni per dispositivi mobili.
Android, iOS e UWP offrono API specifiche del sistema operativo e della piattaforma, a cui gli sviluppatori hanno accesso in C# con Xamarin. Xamarin.Essentials offre una singola API multipiattaforma supportata da qualsiasi applicazione Xamarin.Forms,

Android, iOS o UWP accessibile da codice condiviso, indipendentemente dal modo in cui viene creata l'interfaccia utente [8].

Tramite l’utilizzo di questa libreria è possibile accedere facilmente a funzionalità come l’accelerometro, la georilevazione e la sintesi vocale senza dover richiamare le API specifiche di ciascuna piattaforma.

3. Applicazione SuperMappe con tecnologia Xamarin

3.1 Analisi

SuperMappe è un programma per la realizzazione e visualizzazione di mappe multimediali. Esse sono costituite da nodi, archi che collegano i nodi tra loro e tabelle.
Ciascun nodo appartenente ad una mappa può opzionalmente contenere alcune informazioni di approfondimento come testi, immagini o link visibili su richiesta dell'utente.
L'applicazione presenta principalmente due schermate di interfaccia utente. La prima per consentire la gestione dell'archivio delle mappe e vederne l'anteprima, mentre la seconda per modificare una singola mappa.
Le mappe in uso sono memorizzate nella memoria locale del dispositivo, con la possibilità di salvare le mappe online nello spazio di archiviazione fornito da Google Drive.
Per mantenere la compatibilità con le altre versioni dell'app SuperMappe (SuperMappe EVO e SuperMappeX), le mappe devono essere memorizzate nel formato *SME*.
Il formato SME rappresenta un file zip compresso con estensione proprietaria. Esso contiene i seguenti file [9]:

- properties.ini è il manifesto dei contenuti. Vengono in esso memorizzati in modo testuale i file che compongono l'archivio compresso
- maindoc.graphml è il file che descrive la mappa. È un file di testo in formato GraphML
- text.txt memorizza l'estratto dei testi contenuti nella mappa
- preview.jpg è il file immagine dell'anteprima della mappa
- resources/<nome del nodo o arco>
    - sia i nodi sia gli archi possono contenere la cartella *title*. In essa è presente il file *element.html* che contiene il testo del nodo e le sue caratteristiche grafiche (tipo di font, grandezza del font, margine del testo rispetto ai bordi del nodo, ...)
    - i nodi possono inoltre contenere la cartella *image* e la cartella *deep*. In image è presente l'immagine del nodo (image.png) mentre in deep sono

presenti i contenuti di approfondimento del nodo, come ad esempio i file immagine (1.png, 2.png, ...) e il testo assieme alle sue caratteristiche grafiche (element.html)

L'applicazione SuperMappe permette inoltre di leggere ad alta voce i testi presenti nella mappa tramite la funzionalità di sintesi vocale text-to-speech.

3.2 Progettazione

Di seguito verrà descritta la struttura principale del software SuperMappe.
Per fare ciò ci si servirà di uno schema a blocchi per porre in evidenza le parti più importanti che compongono l'applicazione. Verrà inoltre definito il pattern architetturale che sarà utilizzato per la scrittura del programma. Infine, verrà mostrato il mockup dell'interfaccia grafica che verrà usato come riferimento per la realizzazione del progetto.

3.2.1 Schema a blocchi del progetto

Come viene riportato in Figura 5, l'applicazione viene suddivisa in 4 blocchi principali: l'interfaccia grafica, la rappresentazione grafica del grafo, la persistenza e Google Drive.

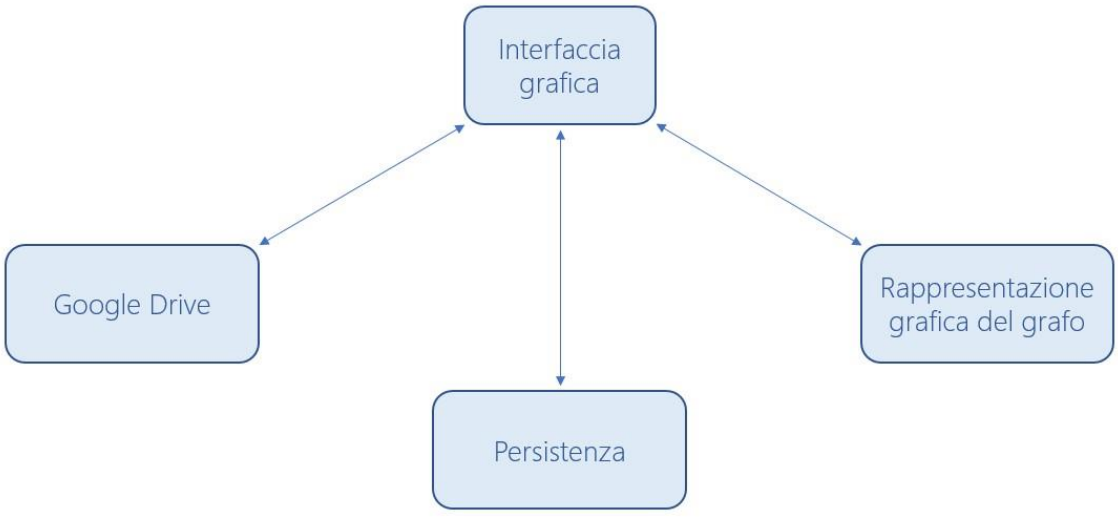

Figura 5: schema a blocchi del progetto SuperMappe

L’interfaccia grafica comprende tutte le schermate dell’applicazione. Tramite esse l’utente può interagire con l’app per effettuare tutte le operazioni di cui necessita. L’interfaccia quindi, è il punto centrale dello schema a cui ogni altro blocco si connette.
Il blocco per la persistenza gestisce la memorizzazione del grafo su memoria persistente. Esso è responsabile per la generazione dei file in formato SME, per la loro lettura e scrittura. Questo blocco si occupa inoltre di trasformare i file SME in dati facilmente gestibili dall’applicazione.
È presente il blocco dedicato alla rappresentazione grafica del grafo. Esso converte l’insieme di dati che compongono il grafo in un insieme di forme e figure. Ad esempio, può disegnare un rettangolo per raffigurare un nodo e una freccia per riprodurre un arco che connetta due nodi tra loro.
Infine, il blocco Google Drive si occupa della gestione e sincronizzazione dei file SME presenti nello spazio cloud di Google.

3.2.2 Pattern Architetturale

Il pattern architetturale che verrà utilizzato in questo progetto è MVVM, ovvero Model-View-ViewModel [10].
Si tratta di un’evoluzione del pattern MVC (Model-View-Controller), ideato da Microsoft ed utilizzato principalmente in applicazioni WPF, UWP e Xamarin.
Come il suo predecessore, si prefigge l’obiettivo di separare la logica di presentazione dei dati dalla logica di business. Per fare ciò, MVVM divide l’applicazione in 3 livelli: Model, View e ViewModel [Figura 6].

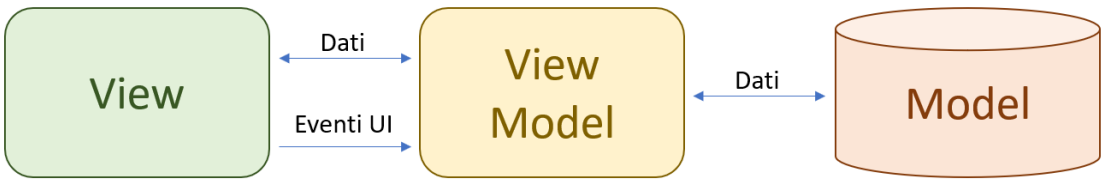

Figura 6: Model-View-ViewModel

Nel Model viene racchiusa la logica di business del programma. È il punto di accesso ai dati come letture da file o da database.

La View rappresenta la vista dell'applicazione, ovvero l'interfaccia grafica che mostrerà i dati.
ViewModel fa da intermediario tra la View e il Model. Esso interagisce con il Model e ne elabora i dati per essere presentati e passati alla View.
La visibilità dei tre livelli è da sinistra verso destra, ovvero ogni livello conosce quelli alla sua destra e ignora quelli alla sua sinistra. Il Model è completamente isolato, quindi non è a conoscenza del ViewModel, tanto meno della View. Il ViewModel non è a conoscenza della View. Ciò permette un disaccoppiamento molto forte, quindi ad esempio è possibile modificare la View senza alcuna ripercussione su ViewModel e Model.
Nonostante ciò, i dati possono fluire in entrambi i versi senza alcuna restrizione. Infatti, il Model potrebbe in un certo istante avere dati più recenti di quelli attualmente in possesso dal ViewModel e visualizzati dalla View. Per questo motivo la View può rimanere in ascolto degli eventi implementati dal ViewModel. Allo stesso modo il ViewModel può rimanere in ascolto degli eventi definiti dal Model. Così facendo si consente una comunicazione bi-direzionale senza influire sulla visibilità che ogni livello ha degli altri.
Il pattern MVVM si rivela molto utile in quanto facilita lo scambio di dati bi-direzionale con l'utilizzo del Data Binding. Per ogni Data Binding, il framework MVVM crea e gestisce gli eventi e handler opportuni senza la necessità di doverlo fare manualmente.
Il Data Binding permette alla View di visualizzare i dati fornitegli dal ViewModel e restare sincronizzata con essi. In questo modo, quando il valore dei dati viene modificato, gli elementi della View ad essi associati riflettono automaticamente le modifiche apportate.

3.2.3 Mockup dell'interfaccia grafica

Il mockup creato si inspira all'interfaccia grafica attualmente in uso da SuperMappeX. L'interfaccia grafica sarà composta principalmente dalle due schermate riportate di seguito in Figura 7.

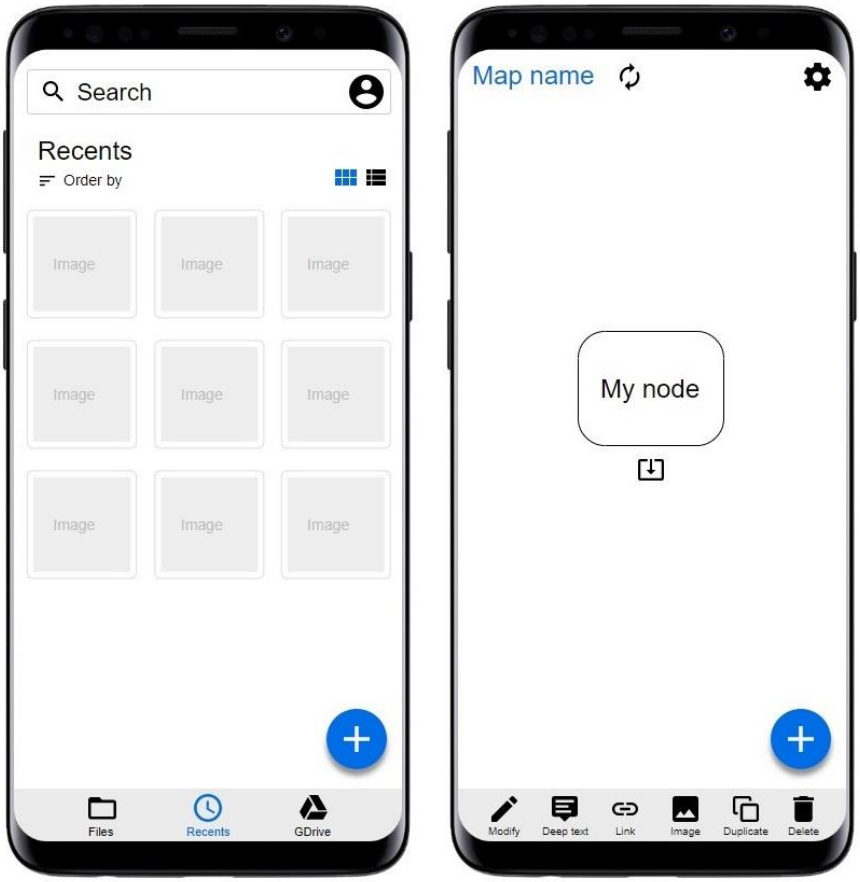

Figura 7: mockup dell'applicazione SuperMappe

La prima schermata viene mostrata all'utente all'avvio dell'app e contiene in primo luogo le mappe aperte di recente. In questa schermata è possibile inoltre effettuare una ricerca testuale sulle proprie mappe ed è presente il pulsante "+" per creare una nuova mappa. È inoltre presente nell'angolo in alto a destra l'icona del profilo dell'utente che indica a quale account Google si è fatto l'accesso.

I pulsanti nella barra inferiore permettono di aprire una mappa memorizzata nel proprio dispositivo, il pulsante per visualizzare le mappe aperte recentemente e quello per aprire una mappa presente all'interno dello spazio cloud Google Drive associato all'account Google.

La seconda schermata viene richiamata all'apertura di una mappa. La gran parte dello spazio a schermo viene dedicato per visualizzare la mappa. È presente in alto il nome della stessa e un'icona per indicare lo stato di sincronizzazione con la sua copia presente in Google Drive. L'icona "ingranaggio" viene utilizzata per richiamare le impostazioni della mappa, come ad esempio rinominare la mappa o modificarne il colore di sfondo.

Alla selezione di un nodo, vengono mostrati all'utente dei pulsanti nella barra sul lato inferiore dello schermo ed un pulsante per poter aggiungere un nuovo nodo collegato a quello selezionato. I pulsanti nella barra inferiore vengono utilizzati per modificare le proprietà del nodo, come ad esempio il suo testo, aggiungere un'immagine o eliminarlo. Infine, è presente nell'angolo inferiore destro il pulsante "+" per aggiungere un nuovo nodo alla mappa.

3.3 Implementazione

Successivamente alla progettazione verrà affrontata l'implementazione dell'applicazione SuperMappe. Verranno descritti nei paragrafi seguenti i principali lavori di implementazione affrontati durante lo sviluppo della stessa.

3.3.1 Ambiente di sviluppo

L'applicazione Xamarin è stata sviluppata utilizzando l'ambiente di sviluppo Visual Studio 2019 installato su un computer Windows. Nello specifico è stata utilizzata l'ultima versione del software uscita nel 2019 con lo scopo di trarre tutti i vantaggi e le novità del linguaggio di programmazione C# 8.0.
La versione C# 8.0 infatti, oltre a presentare una sintassi che consente una migliore leggibilità del codice grazie anche all'introduzione delle *espressioni switch,* permette di scrivere codice maggiormente robusto e con una minor presenza di bug attraverso l'utilizzo dei *tipi di riferimento nullable.* Data la particolare rilevanza dei tipi di riferimento *nullable* nello sviluppo di un'applicazione complessa come quella realizzata e descritta in questo elaborato, ne verranno di seguito descritti i vantaggi.
Tipicamente, nei linguaggi di programmazione orientati agli oggetti una variabile di tipo di riferimento può sempre avere valore nullo. Di conseguenza, nel caso in cui si dovesse dereferenziare erroneamente una variabile il cui valore è nullo, si otterrebbe a tempo di esecuzione un'eccezione di tipo *NullReferenceException*.
Abilitando i tipi di riferimento *nullable* invece, al compilatore viene indicato che tutte le variabili di tipo di riferimento nel codice non possono in alcun modo avere valore nullo.

Nel caso particolare in cui una variabile possa avere valore nullo, questo deve essere esplicitamente indicato aggiungendo il carattere '?' al tipo della variabile [11].
In questo modo il compilatore assiste lo sviluppatore nella scrittura del codice suggerendo di controllare che le variabili che potrebbero essere nulle non abbiano valore nullo prima di dereferenziarle. Inoltre, il compilatore suggerisce di verificare che tutte le variabili che non possono essere nulle siano inizializzate con un valore diverso da nullo.
Di seguito viene riportato un breve esempio di come il compilatore interviene segnalando appositi warning per evitare che errori di scrittura nel codice causino a run-time eccezioni di tipo NullReferenceException.

```
//variabile di tipo di riferimento nullable
Node? n1 = null;
string id = n1.Id; //warning: Dereference of a possibly null reference

//variabile di tipo di riferimento non nullable
Node n2 = null;    //warning: Converting null literal or possible null value to
                   //         non-nullable type

//variabile di tipo di riferimento non nullable
Node n3 = n1;      //warning: Converting null literal or possible null value to
                   //         non-nullable type

//variabile di tipo di riferimento non nullable
Node n4;
if (n1 != null)
    n4 = n1;       //assegnamento correttamente effettuato
```

3.3.2 Modello

La realizzazione del modello è stata implementata sulla base della documentazione fornita dall'azienda Anastasis. Il documento "Descrizione formato SM2-GraphML" [12] fornisce la descrizione di un file con formato *GraphML* utilizzato per memorizzare i nodi e gli archi che costituiscono la mappa. GraphML è un formato di file basato su XML, ovvero un metalinguaggio per la definizione di linguaggi di markup.
Di seguito viene riportata una versione semplificata di file con suddetto formato per mettere in evidenza le sue parti principali.

```
<?xml version="1.0"?>
<graphml>
  <graph>
    <mapprops>
      <mapname><![CDATA[Nuova mappa]]></mapname>
      <author><![CDATA[Federico Gava]]></author>
```

```
    </mapprops>
    <node id="n001">
      <nodedata objtype="rectangle" brushcolor="#FFFFFFFF"
                pencolor="#FF0070C0" penwidth="1"           />
      <geometry x="-1" y="36000" width="12000" height="8000"
                x1="-6001" y1="32000" x2="5999" y2="40000"  />
    </node>
    <node id="n002">
      <nodedata objtype="ellipse" brushcolor="#FFFFFFFF"
                pencolor="#FF0070C0" penwidth="1"        />
      <geometry x="-26451" y="2800" width="12000" height="8000"
                x1="-32451" y1="-1200" x2="-20451" y2="6800"    />
      <titletext><![CDATA[Nodo 2]]></titletext>
    </node>
    <edge id="e001" directed="true" source="n001" target="n002">
      <edgedata type="forward" pencolor="#FF808080" penwidth="1"/>
    </edge>
  </graph>
</graphml>
```

In questo estratto è possibile notare come il file sia composto da 3 elementi principali: *mapprops*, *node* ed *edge*. L'elemento mapprops contiene le proprietà di una mappa come ad esempio il nome della mappa e l'autore. Node è l'elemento che rappresenta un nodo e comprende le sue informazioni estetiche, come forma del nodo e colore dello stesso, in aggiunta alle informazioni sulla sua posizione geometrica all'interno della mappa. Infine, edge rappresenta un arco che connette tra loro due nodi.
Seguendo questo particolare formato di file, è stato creato l'intero modello composto ad esempio dalle classi MapProperties, Node, Edge.

3.3.3 Serializzazione e deserializzazione

Per trasformare un oggetto C# nella sua rappresentazione XML e viceversa, è stata utilizzata la classe *XmlSerializer* [13].
Per poter serializzare e deserializzare è necessario innanzitutto inserire le annotazioni apposite nelle classi del modello.
Di seguito viene riportata parte della classe Node in cui è stata utilizzata l'annotazione *XmlAttribute* nel caso una proprietà di classe fosse un attributo, *XmlElement* nel caso fosse un elemento interno e *XmlIgnore* per ignorare una proprietà.

```
[XmlAttribute("id")]
public override string Id { get; set; }

[XmlElement("nodedata")]
public NodeData { get; set; }
```

```
[XmlElement("geometry")]
public Geometry Geometry { get; set; }

[XmlIgnore]
public string? TitleText { get; set; }

[XmlElement("titletext")]
public XmlCDataSection? XmlTitleText
{
    get
    {
        if (TitleText == null)
            return null;
        else
            return new XmlDocument().CreateCDataSection(TitleText);
    }
    set
    {
        if (value != null)
            TitleText = value.Value;
    }
}
```

La serializzazione e deserializzazione è stata quindi eseguita invocando gli omonimi metodi di un'istanza della classe *XmlSerializer*.

```
//serializzazione
Stream xmlS; //inizializzazione omessa
GraphML g;   //inizializzazione omessa
XmlSerializer serializer = new XmlSerializer(g.GetType());
serializer.Serialize(xmlS, g);

//deserializzazione
Stream xmlS; //inizializzazione omessa
XmlSerializer serializer = new XmlSerializer(typeof(GraphML));
GraphML g = (GraphML)serializer.Deserialize(xmlS);
```

3.3.4 Memorizzazione dei file nel dispositivo

Una volta serializzata la mappa concettuale e generato il file SME, per la sua memorizzazione nel dispositivo si è proceduto come segue.

Le librerie Xamarin.iOS e Xamarin.Android referenziate dall'applicazione Xamarin.Forms includono una versione di .NET specificatamente su misura per queste due piattaforme mobili [14]. Pertanto, è possibile utilizzare la classe *System.IO.File*

direttamente nel progetto condiviso per effettuare le operazioni di lettura e scrittura di file.

Occorre però indentificare il percorso, ovvero la posizione specifica di un elemento all'interno di un file system, dove memorizzare i file. Questo deve essere effettuato diversamente a seconda della piattaforma e per fare ciò è stata utilizzata la classe *DependencyService*.

La classe DepencencyService è un localizzatore di servizi che consente alle applicazioni Xamarin.Forms di richiamare le funzionalità native della piattaforma dal codice condiviso [15].

Il seguente diagramma (Figura 8) mostra com'è possibile invocare una funzionalità nativa all'interno dell'applicazione Xamarin.Forms.

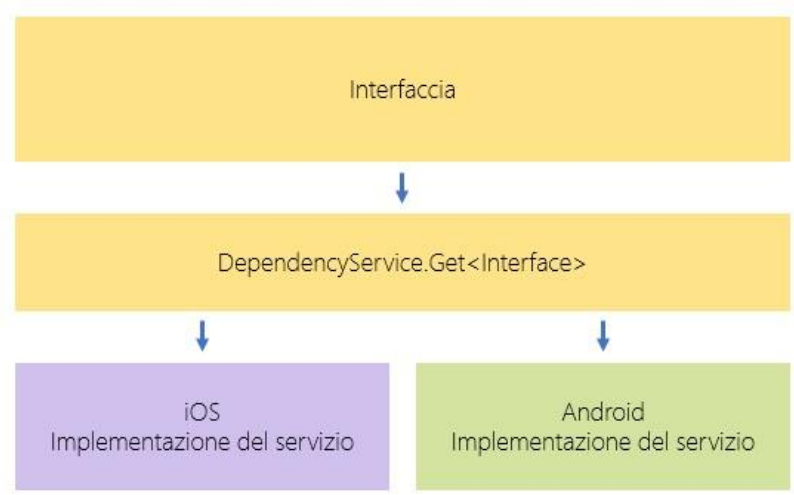

Figura 8: DependencyService

È stata quindi realizzata l'interfaccia *IFileHelper* e nei progetti Android e iOS sono state create rispettivamente le classi *FileHelper_Android* e *FileHelper_iOS*.

Queste due classi devono implementare l'interfaccia inserita nel progetto condiviso e contenere un'annotazione che permetta a Xamarin.Forms di individuare le implementazioni della specifica piattaforma in fase di esecuzione.

Di seguito viene riportato un estratto dell'interfaccia IFileHelper e della classe FileHelper_Android, seguita dall'invocazione del metodo *RootDirectory* nel progetto condiviso.

```
namespace SuperMappe.Model.Persistence
{
    public interface IFileHelper
    {
        public string RootDirectory();
        public Task<PermissionStatus> VerifyAccessToStorageAsync();
    }
}


[assembly: Dependency (typeof(FileHelper_Android))]
namespace SuperMappe.Droid.Model.Persistence
{
    public class FileHelper_Android : IFileHelper
    {
        public string RootDirectory()
        { //implementazione omessa }

        public async Task<PermissionStatus> VerifyAccessToStorageAsync()
        { //implementazione omessa }
    }
}

...

string rootFilesPath = DependencyService.Get<IFileHelper>().RootDirectory();
```

Per quanto riguarda la piattaforma Android, è possibile memorizzare i file nell'archiviazione interna oppure esterna. L'archiviazione interna è una parte del file system a cui è possibile accedere solo dall'applicazione SuperMappe. L'archiviazione esterna invece rappresenta una parte del file system accessibile da qualsiasi applicazione installata nel dispositivo [16].

In Android è stato scelto di memorizzare i file nell'archiviazione esterna in quanto i file presenti nell'archiviazione interna vengono eliminati non appena l'applicazione viene disinstallata.

Per la precisione, i file sono stati memorizzati nella cartella *Documents* facente parte dell'archiviazione esterna e per ottenerne il percorso è stato utilizzato il metodo *GetExternalStoragePublicDirectory* a cui è stato passato come argomento l'enumeratore di tipo *Environment* con valore *DirectoryDocuments*.

```
string path =
Environment.GetExternalStoragePublicDirectory(Environment.DirectoryDocuments)
           .AbsolutePath;
```

L'accesso a cartelle appartenenti all'archiviazione esterna in Android è strettamente regolato.

È necessario inserire nel manifesto dell'applicazione la volontà di accedere a tale tipo di archiviazione.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Inoltre, a partire dalla versione Android 6.0, è necessario un controllo a tempo di esecuzione ogniqualvolta sia necessario accedere ai file presenti nell'archiviazione esterna [17].
È stato realizzato il metodo *VerifyAccessToStorageAsync* per verificare a run-time l'avvenuta autorizzazione da parte dell'utente di questo diritto. Esso inizialmente richiama il metodo *CheckSelfPermission* per verificare se l'utente abbia già concesso l'autorizzazione, successivamente richiama *RequestPermissions* nel caso in cui l'utente non abbia dato il permesso per accedere ai file presenti nell'archiviazione esterna.
In Figura 9 viene riportata la finestra di dialogo visualizzata in seguito all'invocazione del metodo RequestPermission.

Figura 9: richiesta permesso per accedere ai file contenuti nel dispositivo Android

Il metodo *OnRequestPermissionsResult* è il metodo di callback richiamato in seguito alla decisione dell'utente se concedere o meno il permesso.

```
public async Task<PermissionStatus> VerifyAccessToStorageAsync()
{
    if (tcs != null && !tcs.Task.IsCompleted) //reset tcs
    {
        tcs.SetCanceled();
        tcs = null;
    }

    if (!Environment.MediaMounted.Equals(Environment.ExternalStorageState))
```

```
      //! Media mounted and can be read or written to
        return PermissionStatus.Denied;

    if (ContextCompat.CheckSelfPermission(Android.App.Application.Context,
        StoragePermission) == Permission.Granted) //permission already granted
        return PermissionStatus.Granted;

    tcs = new TaskCompletionSource<PermissionStatus>();

    ActivityCompat.RequestPermissions(MainActivity.Instance,
                            new string[] { StoragePermission }, REQUEST_STORAGE);

    return await tcs.Task; //wait for the execution of the
                  //OnRequestPermissionsResult callback (invoked by MainActivity)
}

public static void OnRequestPermissionsResult(int requestCode,
                                  string[] permissions, Permission[] grantResults)
{
    if (requestCode != REQUEST_STORAGE || permissions.Length != 1
        || permissions[0] != StoragePermission)
        return;

    if (tcs == null || tcs.Task.Status == TaskStatus.Canceled)
        return;

    PermissionStatus ps;
    if (grantResults[0] == Permission.Granted)
        ps = PermissionStatus.Granted;
    else
        ps = PermissionStatus.Denied;

    tcs.TrySetResult(ps); //cannot use grantResults[0].
                          //The type Permission derives from Android.
 //FileHelper must work with all platforms! So the type PermissionStatus is used.
}
```

Anche in iOS esiste il concetto di archiviazione interna ed esterna. Nonostante ciò, esistono solo cartelle in archiviazione esterna per memorizzare file di tipo immagine o audio. Data la mancanza della cartella pubblica dove memorizzare i documenti, è stato deciso di memorizzare i file nella cartella di archiviazione interna.
Per ottenere il percorso della cartella *Documents* presente nell'archiviazione interna, è stato invocato il metodo *GetFolderPath* passando *MyDocuments* come parametro d'ingresso [18].

```
string path = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
```

La versione iOS 11 ha introdotto l'app File, un visualizzatore di file per iOS, che consente all'utente di visualizzare e interagire con i file in iCloud e con quelli archiviati da qualsiasi applicazione che la supporti. Per consentire all'utente di accedere direttamente ai file

nell'app, è stata creata una nuova chiave booleana *LSSupportsOpeningDocumentsInPlace* nel file *info.plist* ed impostata con valore *true.*

```
<key>LSSupportsOpeningDocumentsInPlace</key> <true/>
```

Dato che in iOS i file vengono memorizzati nell'archiviazione interna, il permesso per accedere ai file in questo tipo di archiviazione è sempre concesso, quindi il metodo *VerifyAccessToStorageAsync* nel progetto iOS ritorna sempre accesso consentito.

3.3.5 Rappresentazione grafica di una mappa

Per disegnare la mappa all'interno di un canvas è stata usata la libreria *SkiaSharp*.
SkiaSharp è un sistema grafico 2D per .NET e C# basato sul motore grafico *Skia* usato in modo estensivo nei prodotti Google. All'interno di una applicazione Xamarin.Forms, esso permette di disegnare grafica vettoriale 2D, bitmap e testi [19].
Nell'interfaccia grafica destinata alla visualizzazione di una mappa, è stato inserito il canvas *SKCanvasView*.

```
<SKCanvasView x:Name="canvasView" PaintSurface="CanvasView_PaintSurface"/>
```

È stato poi ottenuto un oggetto *SKCanvas* su cui sono stati invocati i metodi *DrawRect*, *DrawRoundRect*, *DrawOval* e *DrawText* per disegnare rispettivamente un rettangolo, un rettangolo con bordi arrotondati, un ovale e del testo. Questi metodi vengono utilizzati per disegnare con aspetti grafici differenti i nodi appartenenti alla mappa concettuale.
Per disegnare un nodo di forma rettangolare viene ad esempio invocato due volte il metodo DrawRect, una prima volta per disegnare il colore di riempimento e la seconda per disegnarne il bordo. Successivamente viene invocato DrawText per disegnare il testo all'interno del nodo.

```
switch (n.NodeData.ObjectType)
{
    case ObjectType.Rectangle:
        c.DrawRect(rectToDraw, backgroundPaint);
        c.DrawRect(rectToDraw, borderPaint);
        DrawText(c, n, rectToDraw);
        break;
```

Nella figura seguente (Figura 10) vengono mostrati tre nodi di forma rettangolare, rettangolare con bordi arrotondati e ovale disegnati con SkiaSharp.

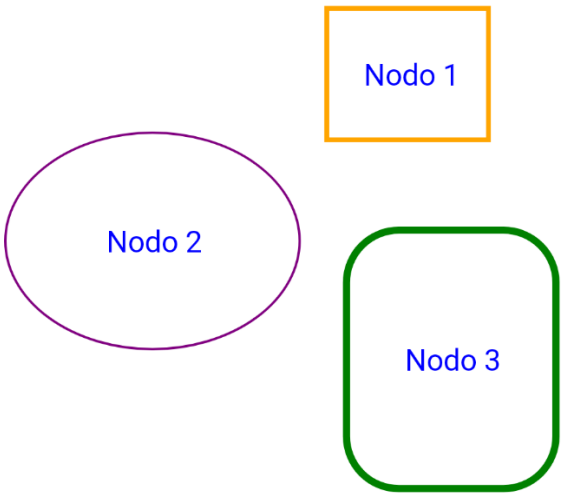

Figura 10: nodi disegnati con SkiaSharp

I nodi così disegnati nel canvas però sono dei semplici pixel colorati. Il canvas non è al corrente che un insieme di pixel in una certa zona dello schermo rappresentano un nodo. È pertanto non immediata la realizzazione della selezione di un nodo, ovvero rilevare il tocco da parte dell'utente di una zona dello schermo dove risiede un nodo.
È necessario inserire nella schermata di visualizzazione della mappa un elemento di tipo *TouchEffect* che rilevi i tocchi dell'utente.

```
<TouchEffect Capture="True" TouchAction="Grid_TouchAction"/>
```

Come riportato dalla documentazione [20], a questo punto si dovrebbe implementare la rilevazione del gesto effettuato dall'utente distinguendolo tra i quattro possibili: tocco, doppio tocco, spostamento o pizzico.
La libreria NuGet *SkiaScene* [21], realizzata da Ondrej Kunc, implementa già questo tipo di rilevazione e quindi è stata utilizzata nel progetto Xamarin.
Dopo aver rilevato il gesto effettuato dall'utente, ciascun nodo verifica se il tocco è stato effettuato su di esso. In caso affermativo, il nodo viene selezionato e vengono mostrati all'utente i comandi per la sua modifica (Figura 11).

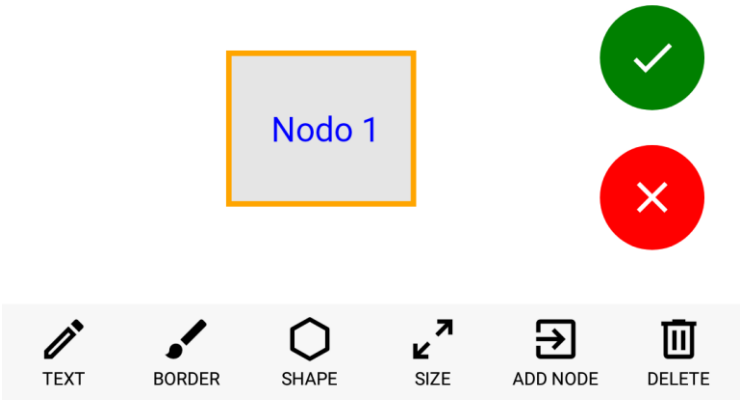

Figura 11: modifica di un nodo

Per quando riguarda il disegno degli archi, è stata creata la classe *Memberships* e *Intersections* per calcolare rispettivamente l'appartenenza di un punto all'interno di una regione del canvas e l'intersezione tra una regione con un segmento.
Ciò è stato utile per identificare precisamente il punto di inizio e quello di fine di ciascun arco. Ogni arco infatti viene disegnato congiungendo il centro del nodo sorgente con il centro del nodo di destinazione. Dal segmento risultante, ne viene mantenuto solo la parte non coperta dagli stessi nodi. Viene riportato nella Figura 12 il segmento iniziale che congiunge i due nodi e il segmento risultate su cui ci si baserà per disegnare l'arco.

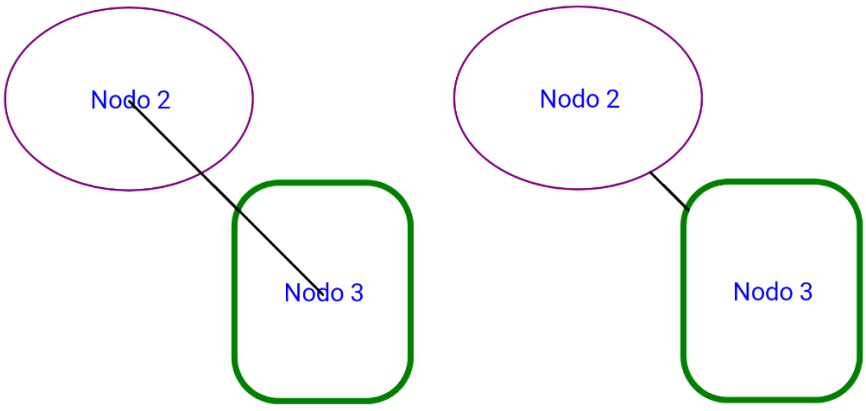

Figura 12: segmento che congiunge i due nodi (a sinistra) e il segmento risultante (a destra)

Ottenute le coordinate iniziali e finali del segmento, esso è stato disegnato invocando la funzione *DrawLine* del SKCanvas. Infine, sono state disegnate le frecce dell'arco creando un *SKPath* ed invocando il metodo *DrawPath*.

```
c.DrawLine(sSKPoint, tSKPoint, paint);   //edge

SKPath arrow = new SKPath();
arrow.MoveTo(0, 0);
arrow.RLineTo(15, 12);
arrow.RLineTo(0, -24);
arrow.Close();

//arrow of the edge in source side
if (e.EdgeData.Type == EdgeType.Backward || e.EdgeData.Type == EdgeType.Both)
{
    c.Save();
    c.Translate(sSKPoint);
    //(0,0) coordinate is now the starting point of the edge
    float beginAngle = (float)Math.Atan2(tSKPoint.Y - sSKPoint.Y,
                                         tSKPoint.X - sSKPoint.X);
    c.RotateRadians(beginAngle);
    //the edge is horizontal and the ending point in on the positive side of X
    //axis (eg. (+X,0) )
    c.DrawPath(arrow, paint);
    c.Restore();
}
```

In Figura 13 viene riportato il disegno finale dell'arco.

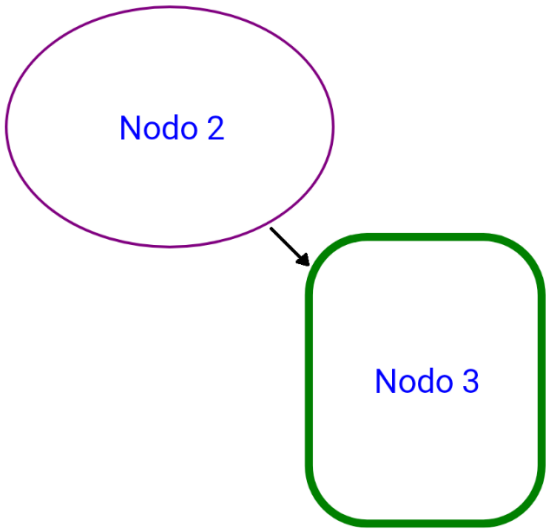

Figura 13: disegno dell'arco all'interno del canvas

Similarmente a quando fatto per i nodi, ciascun arco verifica se l'utente l'abbia toccato. In caso affermativo, viene mostrata all'utente la lista dei controlli su cui può agire per modificare le caratteristiche dell'arco selezionato (Figura 14).

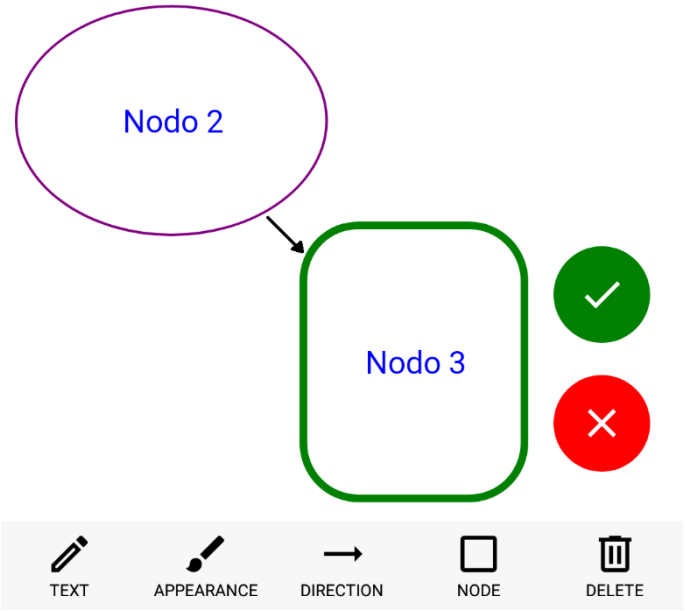

Figura 14: modifica di un arco

3.3.6 Memorizzazione delle mappe in Google Drive

Per poter effettuare l'upload dei file SME presenti sul dispositivo nello spazio cloud Google Drive, tipicamente diverse soluzioni sono percorribili ma solo una è implementabile in un progetto Xamarin.

Google fornisce una libreria per ambiente .NET dove è possibile, all'interno di codice C#, richiamare i metodi per accedere ai servizi Google come Google Drive, Google Maps e Calendar. Purtroppo, questa libreria attualmente è in stato di sola manutenzione e non ha il supporto per molti framework utilizzati al giorno d'oggi tra cui Xamarin [22].

Un'altra via percorribile è quella di richiamare dal progetto Xamarin.Android le API della libreria Google Drive presente nell'SDK di Android. Sebbene permetterebbe di caricare un file in Google Drive con la semplice invocazione di un metodo, ciò non potrebbe essere utilizzato nella parte di progetto relativa ad iOS.

L'unica alternativa possibile è l'accesso ai servizi Google tramite l'uso di API REST [23]. Le API REST si basano su HTTP. Il loro funzionamento prevede una struttura URL, ben definita atta a indicare univocamente una risorsa, e l'uso di verbi HTTP specifici per il recupero di informazioni (GET) o per la loro modifica (POST, PUT, PATCH, DELETE) [24].

Prima che sia possibile accedere al servizio Google Drive, è necessario effettuare l'autenticazione. Per fare ciò è stato utilizzato il protocollo OAuth 2.0.

OAuth è uno standard aperto per l'autenticazione e consente al proprietario di una risorsa di notificare al fornitore della stessa che deve essere concessa l'autorizzazione a terze parti per accedere alle informazioni senza condividere l'identità del detentore della risorsa [25].

L'autenticazione segue molteplici fasi visibili nella Figura 15 e descritte nelle righe seguenti.

Inizialmente l'applicazione che vuole accedere alla risorsa dell'utente deve aprire la pagina web di autenticazione dell'*identity provider* (es. Google, Microsoft, Facebook e Twitter) (fasi 1,2).

L'identity provider gestisce l'autorizzazione verificando se i dati di accesso inseriti dall'utente nella pagina web corrispondono ad un valido account e, in caso affermativo, ritorna all'applicazione un codice di autorizzazione (fasi 3,4).

L'applicazione in questa fase scambia con l'identity provider il codice di autorizzazione per un token di accesso (fasi 5,6).

Infine, l'applicazione può utilizzare il token di accesso per utilizzare le API rese disponibili dall'identity provider (fasi 7,8).

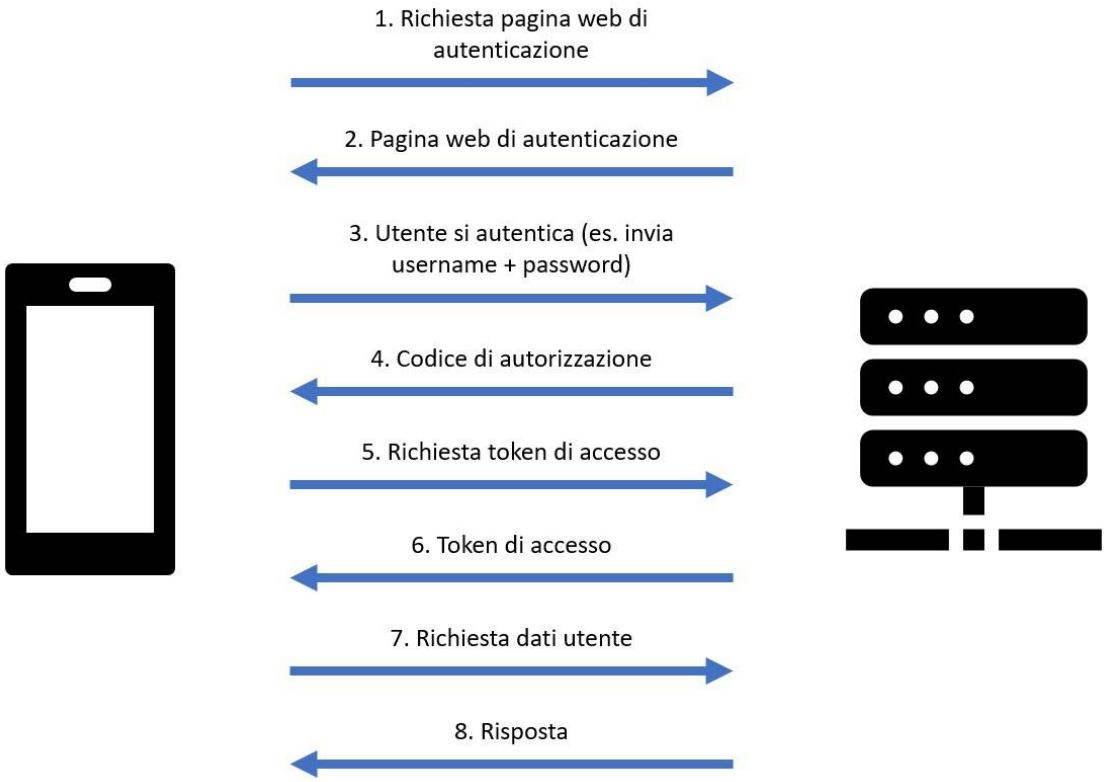

Figura 15: fasi protocollo OAuth 2.0

Per abilitare l'autenticazione OAuth, dalla console di sviluppo Google (console.developers.google.com) sono stati creati due *clientID* rispettivamente per la versione Android e per la versione iOS dell'applicazione SuperMappe.

È stata utilizzata la libreria NuGet *Xamarin.Auth* [26] per avviare il protocollo OAuth e ricevere il token di accesso dall'identity provider. Nello specifico è stato inizializzato un oggetto *OAuth2Authenticator* con *clientID* e *googleDriveFullAccessScope* come parametri d'ingresso, quest'ultimo ad indicare che il token sarà utilizzato per accedere al servizio Google Drive.

```
const string googleDriveFullAccessScope = "https://www.googleapis.com/auth/drive";
const string authorizeUrl = "https://accounts.google.com/o/oauth2/v2/auth";
const string accessTokenUrl = "https://www.googleapis.com/oauth2/v4/token";
private readonly TaskCompletionSource<AuthenticatorCompletedEventArgs> tcs;

public GoogleOAuthAuthentication()
{
    string clientID = DependencyService.Get<ICredentials>().ClientID();
    string redirectUrl = DependencyService.Get<ICredentials>().RedirectUrl();

    tcs = new TaskCompletionSource<AuthenticatorCompletedEventArgs>();
```

```
    Uri authorizeUri    = new Uri(authorizeUrl);
    Uri redirectUri     = new Uri(redirectUrl + ":/oauth2redirect");
                                        //reverse  clientID + ":/oauth2redirect"
    Uri accessTokenUri  = new Uri(accessTokenUrl);

    OAuth2Authenticator authenticator = new OAuth2Authenticator(clientID, null,
                                                        googleDriveFullAccessScope,
                                                        authorizeUri, redirectUri,
                                                        accessTokenUri, null, true);
}
```

In seguito all'inizializzazione di una variabile di tipo *GoogleOAuthAuthentication*, è possibile invocare il metodo *Login*. Esso richiama il metodo *Login* di un oggetto di classe *OAuthLoginPresenter* per mostrare a schermo la pagina web dove l'utente può inserire i dati d'accesso. Successivamente viene atteso l'esito dell'operazione con il metodo di callback *OnAuthenticatorCompletedEventArg*.

```
public async Task<AuthenticatorCompletedEventArgs> Login()
{
    StartLogin();
    return await tcs.Task;//wait for OnAuthenticatorCompleted
}

private void StartLogin()
{
    OAuth2Authenticator auth = AuthenticationState.Authenticator;

    auth.Completed += OnAuthenticatorCompleted;

    OAuthLoginPresenter presenter = new OAuthLoginPresenter();
    presenter.Login(auth); //invoke the login presenter
}

private void OnAuthenticatorCompleted(object sender,
                                      AuthenticatorCompletedEventArgs eventArgs)
{
    if (sender is OAuth2Authenticator authenticator)
        authenticator.Completed -= OnAuthenticatorCompleted;

    tcs.TrySetResult(eventArgs);
}
```

In Figura 16 è possibile notare come si presenta la schermata di autenticazione e la richiesta del permesso per accedere ai dati contenuti in Google Drive.

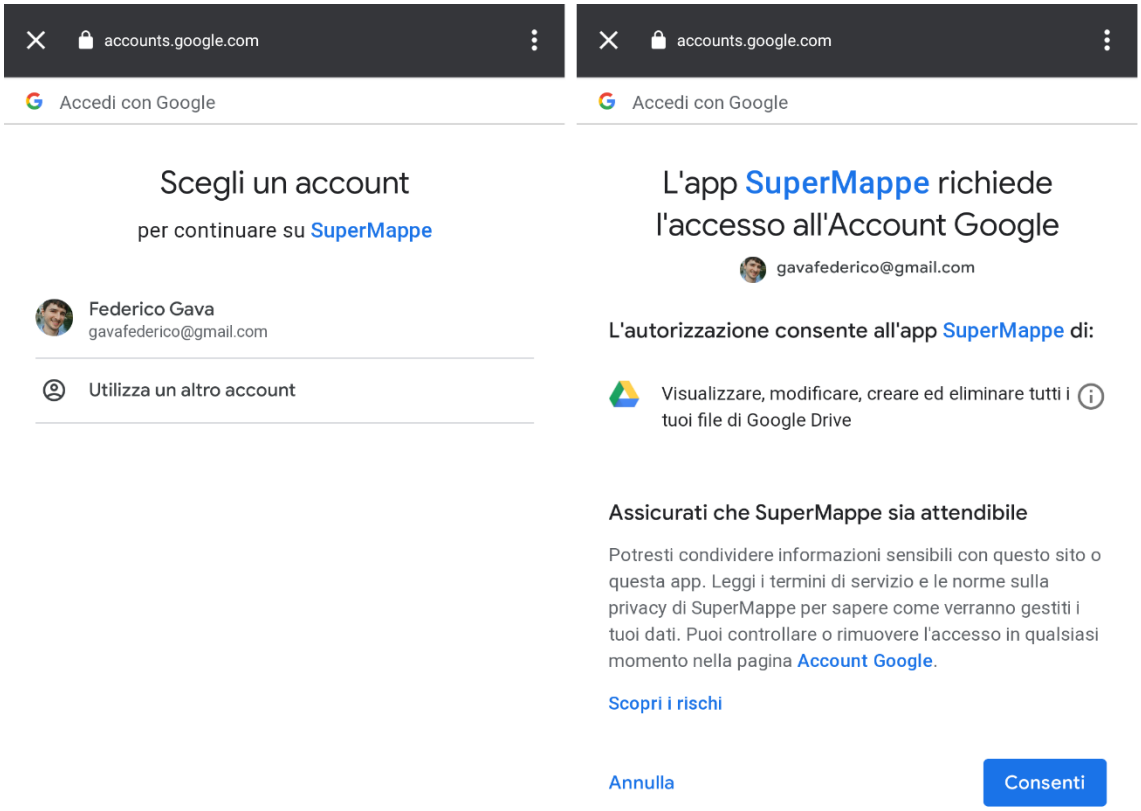

Figura 16: login account Google (a sinistra) e richiesta permesso di accesso ai dati in Google Drive (a destra)

Ottenuto il token di accesso, è stata inizializza una variabile di classe *GoogleDriveRESTService* con lo scopo di creare una istanza di *HttpClient* contenente il token come parametro di autorizzazione.

```
public GoogleDriveRESTService(string accessToken)
{
    client = new HttpClient();
    client.DefaultRequestHeaders
          .Authorization = new AuthenticationHeaderValue("Bearer", accessToken);
}
```

In questa classe sono stati implementati diversi metodi. I più rilevanti, in seguito descritti, vengono utilizzati per ottenere la lista di file SME presenti in Google Drive, scaricare un file presente in Google Drive, caricarne uno presente nel dispositivo su cui è in esecuzione l'app SuperMappe o eliminare un file dallo spazio cloud.

Per ciascuno di questi è necessario specificare l'URI corrispondente alla richiesta da eseguire. Ad esempio per richiedere la lista di tutti i file SME presenti nello spazio cloud

di Google è necessario utilizzare l'URI seguente: https://www.googleapis.com/drive/v3/files?q=mimeType=application/supermappe.

Successivamente è possibile utilizzare la variabile client di tipo HttpClient per invocare una richiesta HTTP del tipo compatibile con l'azione da eseguire. Nel caso citato precedentemente, dove viene richiesta la lista di file SME presenti su Google Drive, la richiesta HTTP da effettuare dovrà essere di tipo GET.

Effettuata la richiesta, è possibile leggere la variabile di ritorno di tipo *HttpResponseMessage* per verificare l'esito dell'operazione ed in caso positivo, consultarne il risultato.

Google Drive assegna per ogni file un codice identificativo. In questo modo è possibile caricare nella stessa directory molteplici file con lo stesso nome. L'univocità di un file all'interno di Google Drive infatti, non è data dal suo percorso e nome, come accade comunemente nei file system, ma dal suo ID.

Nella lettura dei file SME presenti in Google Drive viene quindi letto per ciascun file il suo nome ed il suo identificativo.

Nel codice che segue, viene riportata la richiesta di elencare i file SME presenti in Google Drive.

```
public async Task<Dictionary<string, string>> GetAllSMEFileIDsAndFilenames()
{
    Dictionary<string, string> result = new Dictionary<string, string>();

    Uri uri = new Uri(https://www.googleapis.com/drive/v3/files?q=mimeType='
                                                        + SmeMIMEMediaType + "'");

    HttpResponseMessage response = await client.GetAsync(uri);

    if (response.IsSuccessStatusCode)
    {
        string content = await response.Content.ReadAsStringAsync();
        FileListJSON fileListJSON = JsonConvert
                                            .DeserializeObject<FileListJSON>(content);

        foreach (FileJSON file in fileListJSON.Files)
            result.Add(file.Id, file.Name);
    }
    return result;
}
```

Il codice ID è utilizzato dalle API Google Drive per effettuare azioni sul file come la sua modifica, la sua eliminazione o il suo download nel dispositivo.

Similarmente a quanto visto per la richiesta della lista di file SME presenti in Google Drive, il download degli stessi viene eseguito specificando l'URI ed invocando il metodo *GetAsync*. Nella risposta è contenuto il file scaricato, il quale viene copiato nel dispositivo.
Di seguito viene riportato il codice del metodo utile al download di un file seguita dall'invocazione del metodo stesso.

```
public async Task<HttpResponseMessage> DownloadSMEFile(string fileID)
{
    Uri uri = new Uri(https://www.googleapis.com/drive/v3/files/
                      + fileID + "?alt=media");

    return await client.GetAsync(uri);
}


...


HttpResponseMessage response = await DownloadSMEFile(file.Key);

if (response.IsSuccessStatusCode)
{
    using FileStream fileStream = new FileStream(rootPath + "/" + file.Value,
                                                 FileMode.Create,
                                                 FileAccess.Write,
                                                 FileShare.None);
    await response.Content.CopyToAsync(fileStream);
}
```

L'upload di un file viene eseguito creando una variabile di tipo *MultipartContent* contenente il nome del file ed il file stesso, seguita dall'invocazione del metodo *PostAsync*.

```
//not idempotent: invoking this method multiple times results in having on GDrive
//more copy of the same file with the same filename but different fileID
public async Task<HttpResponseMessage> UploadNewSMEFile(string path)
{
    Uri uri = new
Uri("https://www.googleapis.com/upload/drive/v3/files?uploadType=multipart");

    string fileName = Path.GetFileName(path);

    using MultipartContent multipartC = new MultipartContent("related",
                                                             "thisIsTheBundary");

    string json = "{\"name\": \""+fileName+"\"}";
    StringContent jsonContent = new StringContent(json, Encoding.UTF8,
                                                  "application/json");

    multipartC.Add(jsonContent);
```

```
    ByteArrayContent fileContent = new ByteArrayContent(File.ReadAllBytes(path));
    fileContent.Headers.ContentType = new MediaTypeHeaderValue(SmeMIMEMediaType);

    multipartC.Add(fileContent);

    return await client.PostAsync(uri, multipartC);
}
```

Infine, l'eliminazione di un file presente in Google Drive è stata effettuata richiamando il metodo *DeleteAsync* utilizzando l'URI contenente l'ID del file da eliminare.

```
public async Task<HttpResponseMessage> DeleteFile(string fileId)
{
    Uri uri = new Uri("https://www.googleapis.com/drive/v3/files/" + fileId);
    return await client.DeleteAsync(uri);
}
```

3.3.7 Xamarin Essentials

Come menzionato nel capitolo 3.1 Analisi, l'applicazione SuperMappe su cui si basa questa tesi, utilizza inoltre la funzionalità di text-to-speech per leggere tramite un sintetizzatore vocale il testo contenuto nelle mappe concettuali.
A questo scopo è stata inserita nel progetto la libreria Xamarin.Essentials descritta nel capitolo 2. Tecnologia Xamarin.
Xamarin.Essentials offre API multipiattaforma per utilizzare funzionalità comuni ai dispositivi mobili come la geolocalizzazione e l'accelerometro. Tra questi vi è inoltre la funzionalità text-to-speech.
Per poter utilizzare questa libreria all'interno di un progetto Xamarin, nel progetto Android è necessario effettuare la sua inizializzazione. Nello specifico deve essere invocato il metodo *Init* all'avvio dell'applicazione.

```
protected override void OnCreate(Bundle savedInstanceState)
{
    [...]
    Xamarin.Essentials.Platform.Init(this, savedInstanceState);
```

Fatto ciò è possibile invocare il metodo *SpeakAsync* all'interno del progetto Xamarin.Forms per leggere il testo passato come argomento.

```
await TextToSpeech.SpeakAsync(text);
```

3.4 Collaudo

L'applicazione fin qui sviluppata è stata collaudata nella piattaforma Android ed in iOS. L'applicazione SuperMappe è stata testata nella sua variante Android mediante l'utilizzo del mio personale smartphone Moto G5 Plus con installato il sistema operativo Android 8.1.

Vengono riportate in Figura 17 le due schermate principali dell'app ottenute con il dispositivo Android, le quali riportano rispettivamente l'elenco di tutte le mappe presenti nel dispositivo e la visualizzazione e modifica di una singola mappa.

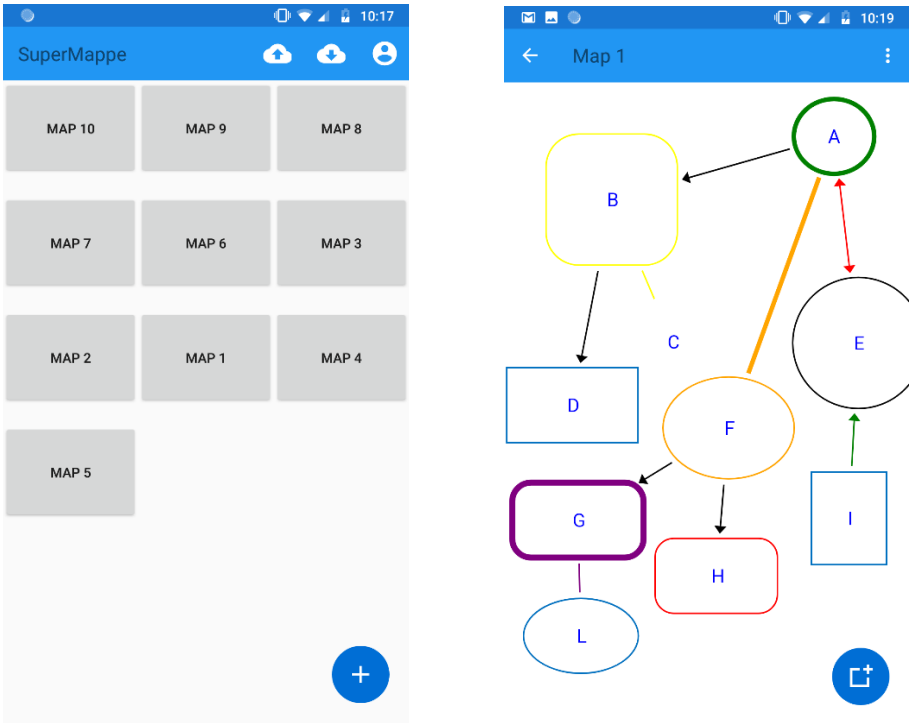

Figura 17: elenco di tutte le mappe (a sinistra) e visualizzazione e modifica di una mappa (a destra) in Android

Per quanto riguarda invece la versione iOS, la compilazione del progetto Xamarin deve obbligatoriamente essere effettuata dal software Xcode installato in un dispositivo Mac.

È stato quindi installato Xcode e Visual Studio for Mac in un dispositivo Mac Mini fornito dall'Università. Tuttavia, l'ambiente di sviluppo Xcode non è stato direttamente utilizzato in quanto è stata sfruttata la funzionalità di Visual Studio *Associa a Mac*, che ha permesso al PC Windows di connettersi automaticamente al Mac per effettuare la compilazione del progetto.

Conclusa la compilazione è stato possibile testare l'applicazione in versione iOS tramite uno dei simulatori di dispositivi iOS forniti dal Mac stesso. Nello specifico è stato scelto di testare l'app SuperMappe con il dispositivo iPhone Xs Max.

Analogamente a quanto mostrato dalla Figura 17, in Figura 18 vengono riportate le due schermate dell'applicazione ottenute con il simulatore di iPhone Xs Max.

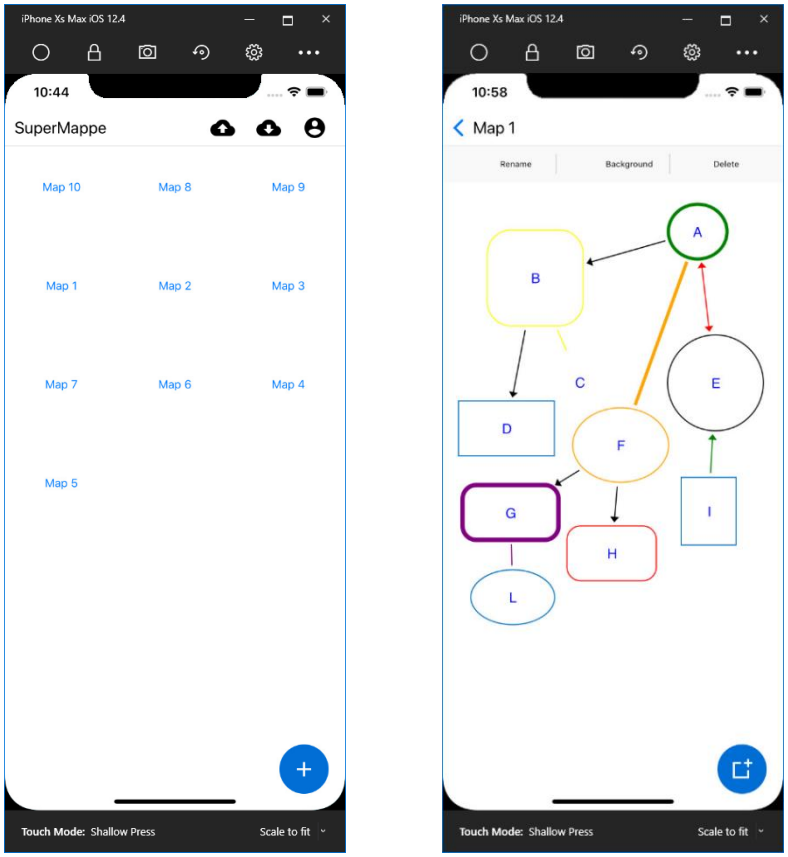

Figura 18: elenco di tutte le mappe (a sinistra) e visualizzazione e modifica di una mappa (a destra) in iOS

Sono stati effettuati alcuni test per valutare le prestazioni dell'app SuperMappe nelle versioni Android e iOS.

Nello specifico è stato misurato il consumo di memoria RAM ed il tempo di avvio dell'applicazione mentre erano presenti nel dispositivo 10 mappe costituite ciascuna da 10 nodi e 9 archi.

Il consumo di RAM è stato ottenuto tramite l'invocazione del metodo *PrivateMemorySize64* della classe *Process*, il quale misura la quantità di memoria privata, in byte, allocata per il processo associato [27].

Il tempo di avvio invece è stato misurato tramite l'inizializzazione di una variabile di classe *Stopwatch* e l'invocazione dei metodi *Start* ed *Elapsed* rispettivamente prima e al completamento dell'inizializzazione dell'applicazione.

In Figura 19 sono stati riportati i risultati ottenuti dai test.

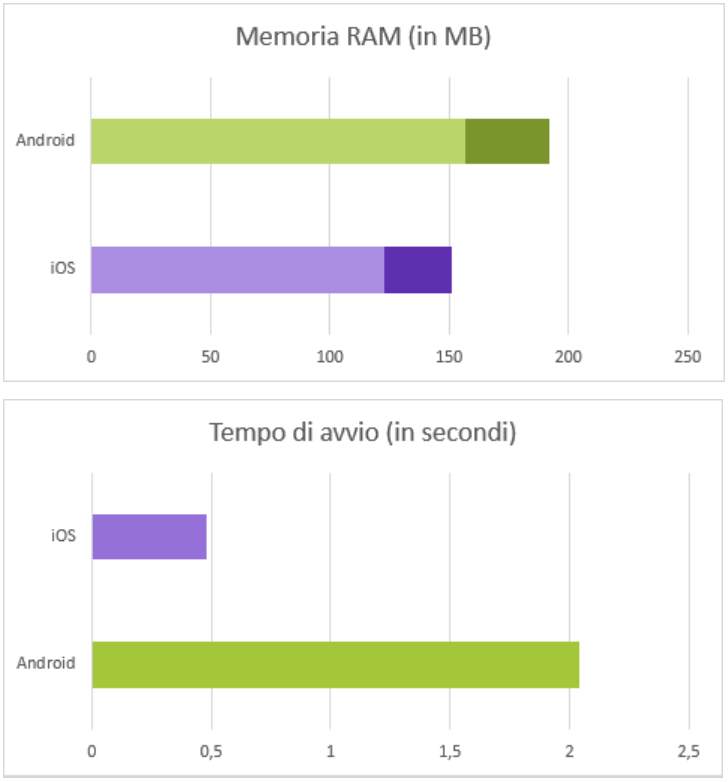

Figura 19: memoria RAM occupata (sopra) e tempo di avvio (sotto) in Android e iOS

È stata osservata un'occupazione di memoria RAM similare nelle due piattaforme. Durante la visualizzazione dell'elenco delle mappe sono stati ottenuti rispettivamente 157 MB di memoria RAM occupata in Android e 123 MB in iOS. Dopo aver aperto una delle mappe presenti nel dispositivo, si è ottenuta un'occupazione di RAM di 192 MB in Android e 151 MB in iOS.

Durante la misurazione del tempo di avvio dell'applicazione, invece, nelle due piattaforme sono stati ottenuti risultati molto differenti. In Android l'applicazione si è avviata in 2,04 s mentre in iOS in 0,48 s. La differenza riscontrata potrebbe essere conseguenza diretta dell'uso del simulatore fornito dal Mac Mini e quindi della grande disparità tra la potenza computazionale del Mac rispetto a quella dello smartphone Android.

Le applicazioni eseguite in entrambi i sistemi operativi non hanno manifestato alcun problema, infatti sono risultate fluide nel loro utilizzo senza impuntamenti.

4. Conclusioni e sviluppi futuri

Il presente lavoro ha lo scopo di descrivere la realizzazione dell'applicazione SuperMappe utilizzando la tecnologia Xamarin per rendere disponibile questo programma nelle piattaforme Android e iOS.

In questo elaborato sono stati descritti i punti principali che hanno costituito l'implementazione del progetto, come ad esempio il modo in cui serializzare le classi del modello, la creazione di file con estensione SME, la visualizzazione di una mappa e l'accesso allo spazio cloud Google Drive dell'utente.

A lavoro compiuto è stato osservato come l'obbiettivo che si prefigge la tecnologia Xamarin, ovvero permettere agli sviluppatori di realizzare applicazioni native condividendo il codice su diverse piattaforme, sia stato raggiunto.

Il codice condiviso tra le diverse piattaforme ha raggiunto le 6257 righe a fronte della scrittura di codice specifico per le piattaforme Android e iOS rispettivamente di 210 e 136 righe. Nella Figura 20 si può osservare un grafico a torta, il quale mostra come il codice condiviso tra i diversi progetti abbia raggiunto il 95% del codice totale.

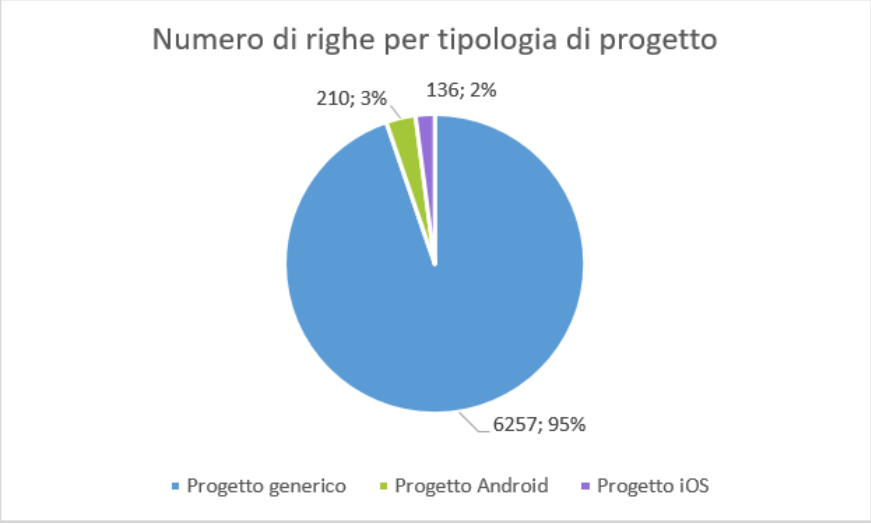

Figura 20: numero di righe per tipologia di progetto

Lo studio prodotto ha permesso la realizzazione della gran parte delle funzionalità presenti nel software di proprietà dell'azienda Anastasis.

Un possibile ampliamento futuro del progetto fin qui descritto potrebbe consistere nell'estensione del progetto Xamarin ad un'ulteriore piattaforma, come ad esempio UWP (Universal Windows Platform).

A fronte di limitate modifiche infatti, SuperMappe potrebbe essere resa compatibile con tutti i dispositivi Windows 10 presenti sul mercato, tra cui computer fissi, portatili e tablet.

A conclusione di questo lavoro di tesi, il codice sviluppato verrà consegnato all'azienda Anastasis. L'azienda potrà così analizzare il progetto valutando anche la possibilità di applicare la tecnologia Xamarin non a Supermappe, ma in generale a tutti i loro prodotti software, come ad esempio ePico!, MateMitica e Geco.

5. Bibliografia

[1] Rivoluzione digitale
https://it.wikipedia.org/wiki/Rivoluzione_digitale

[2] Anastasis Società Cooperativa Sociale
https://www.anastasis.it/chi-siamo/

[3] SuperMappeX
https://www.anastasis.it/catalogo-generale/supermappex-educational/

[4] Xamarin
https://it.wikipedia.org/wiki/Xamarin

[5] PDF: SuperMappeX - Guida introduttiva

[6] Che cos'è Xamarin?
https://docs.microsoft.com/it-it/xamarin/cross-platform/get-started/introduction-to-mobile-development

[7] Creating Mobile Apps with Xamarin.Forms – Capitolo 1
https://xamarin.azureedge.net/developer/xamarin-forms-book/XamarinFormsBook-Ch01-Apr2016.pdf

[8] Xamarin.Essentials
https://docs.microsoft.com/it-it/xamarin/essentials/get-started?context=xamarin%2Fandroid&tabs=windows%2Candroid

[9] PDF: SuperMappe EVO – Generale

[10] Model-View-ViewModel
https://docs.microsoft.com/it-it/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

[11] Tipi di riferimento nullable
https://docs.microsoft.com/it-it/dotnet/csharp/nullable-references

[12] Descrizione formato SM2-GraphML

[13] XmlSerializer
https://docs.microsoft.com/it-it/dotnet/api/system.xml.serialization.xmlserializer?view=netframework-4.8

[14] Creating Mobile Apps with Xamarin.Forms – Capitolo 20
https://download.xamarin.com/developer/xamarin-forms-book/XamarinFormsBook-Ch20-Apr2016.pdf

[15] Depencency Service
https://docs.microsoft.com/it-it/xamarin/xamarin-forms/app-fundamentals/dependency-service/introduction

[16] Archiviazione file e accesso con Xamarin.Android
https://docs.microsoft.com/it-it/xamarin/android/platform/files/

[17] Archiviazione esterna
https://docs.microsoft.com/it-it/xamarin/android/platform/files/external-storage?tabs=windows

[18] Accesso al file system con Xamarin.iOS
https://docs.microsoft.com/it-it/xamarin/ios/app-fundamentals/file-system

[19] SkiaSharp
https://docs.microsoft.com/it-it/xamarin/xamarin-forms/user-interface/graphics/skiasharp/

[20] Manipolazioni tramite tocco
https://docs.microsoft.com/it-it/xamarin/xamarin-forms/user-interface/graphics/skiasharp/transforms/touch

[21] Libreria SkiaScene
https://github.com/OndrejKunc/SkiaScene

[22] Google APIs client Library for .NET
https://github.com/googleapis/google-api-dotnet-client

[23] Introduction to Google Drive API
https://developers.google.com/drive/api/v3/about-sdk

[24] Representational State Transfer (REST)
https://it.wikipedia.org/wiki/Representational_State_Transfer

[25] Autenticare gli utenti con un provider di identità
https://docs.microsoft.com/it-it/xamarin/xamarin-forms/data-cloud/authentication/oauth

[26] Xamarin.Auth
https://github.com/xamarin/Xamarin.Auth

[27] Process.PrivateMemorySize64
https://docs.microsoft.com/it-it/dotnet/api/system.diagnostics.process.privatememorysize64?view=netframework-4.8