Università Politecnica delle Marche

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica e dell'Automazione

Tesi di Laurea

Progettazione e realizzazione di un'app Xamarin per la gestione dei clienti di un ristorante

Design and realization of a Xamarin app for the management of clients in a restaurant

Relatore

Prof. Domenico Ursino

Candidato

Edoardo Russo

Anno Accademico 2019-2020

Indice

Elenco delle figure

Listings

Introduzione

Lo smartphone al giorno d'oggi è diventato un oggetto così utile che è difficile immaginare la nostra vita quotidiana senza di esso.

Secondo le statistiche più recenti, negli ultimi anni il numero di smartphone in circolazione è aumentato da 2.5 miliardi (circa il 33.58% della popolazione mondiale) fino ad arrivare nel 2020, a 3.5 miliardi (circa il 44.98% della popolazione mondiale).

Il mondo degli smartphone è in continua crescita e innovazione e, con esso, anche altre tecnologie (come l'Internet of Things, dispositivi wearables, la realtà aumentata, 5G, etc.) che contribuiscono a far crescere l'attenzione su questo mondo creando un vero e proprio ecosistema oggetto di studio in tutto il mondo.

Si stima che, nel 2018, i ricavi generati dalle app, in tutto il mondo, fossero superiori a 365 miliardi di dollari, supponendo che l'odierno tasso di crescita si mantenga costante, si ipotizza che nel 2023 si potrebbero avere più di 935 miliardi di dollari di ricavi da app a pagamento e dalle pubblicità in-app [4].

Non si può parlare di sviluppo mobile senza considerare il mercato degli smartphone in relazione ai sistemi operativi preinstallati. Due sono i principali sistemi, ovvero Android, con l'87% del mercato, e iOS, con il 13%.

Negli ultimi anni, salvo alcuni tentativi da parte di altre aziende, le percentuali si sono mantenute all'incirca costanti, portando Android ed iOS ad essere, effettivamente i dominatori del mercato.

Risulta, quindi, importante chiedersi quali siano le tecnologie che il mercato propone per lo sviluppo mobile; in particolare, risulta interessante scegliere la tecnologia più adatta per arrivare al maggior numero di utenti riducendo i costi di sviluppo. Attualmente è possibile classificare le app in tre categorie principali:

- app native;
- progressive web app;
- app cross-platform.

Le app native sono applicazioni sviluppate per un sistema operativo specifico. Questa scelta porta a realizzare un'app perfettamente integrata con il sistema operativo, sfruttando in maniera ottimale le risorse hardware e software che quest'ultimo fornisce.

Di contro, le app native avranno costi maggiori, dovendo sviluppare due app singole, e avranno problemi di portabilità, essendo ogni app strettamente legata al sistema operativo per cui è stata pensata.

Le progressive web app rappresentano, di fatto, un ibrido tra le app native e le classiche pagine Internet. Questo approccio, sfruttando le tecnologie proprie dello sviluppo web, permette di sviluppare app compatibili con entrambi i sistemi operativi mobile, consentendo, inoltre, di utilizzare al meglio le risorse del dispositivo mobile.

In questo modo sarà possibile abbattere i costi e i tempi di sviluppo a discapito di prestazioni inferiori e di un'estetica meno fedele a quella propria dei sistemi operativi mobile.

Le app cross-platform uniscono i vantaggi di entrambe le alternative precedentemente descritte. Esse permettono di sviluppare applicazioni su entrambi i sistemi operativi mobile, riutilizzando il codice condiviso, ma si distinguono dalle progressive web app per prestazioni e interfaccia utente. Le app cross-platform avranno aspetto e performance del tutto paragonabili a quelle delle app native.

I framework principali di questa categoria sono React Native, Flutter e Xamarin; essi, pur condividendo la filosofia generale, si basano su tecnologie diverse.

Xamarin è un framework open-source basato su C#, pensato per consentire lo sviluppo di applicazioni mobile cross-paltform. Creato da Miguel De Icaza nel 2011, raccoglie l'eredità del progetto Mono sul quale si basa.

Attraverso l'uso del linguaggio C# e della piattaforma .NET, Xamarin consente lo sviluppo di applicazioni Android e iOS con la possibilità di utilizzare tutte le caratteristiche native, sia per quanto riguarda l'interfaccia che le funzionalità tipiche, dei diversi sistemi operativi.

Xamarin è completamente integrato con Visual Studio, IDE che fornisce tutti gli strumenti avanzati che possono affiancare il lavoro del programmatore e non solo.

Xamarin è stato utilizzato per implementare un'app semplice e intuitiva che permettesse di gestire alcune delle funzionalità richieste dalle attività di un ristorante.

L'obiettivo di quest'app è quello di migliorare l'esperienza dei clienti di un ristorante, portando molti vantaggi anche a chi gestisce questo tipo di attività.

L'attività dell'utente sarà migliorata perché, interagendo con quest'app, egli potrà:

- usufruire del menù offerto dal ristorante in maniera più veloce e intuitiva;
- ordinare le pietanze proposte dal ristorante;
- fare una richiesta di prenotazione di un tavolo, in maniera più veloce senza dover necessariamente contattare il gestore dell'attività;
- effettuare i pagamenti direttamente tramite l'app;
- visualizzare le informazioni e i contatti del ristorante.

L'app promette di velocizzare e semplificare molte delle attività che si svolgono all'interno di un ristorante, consentendo al gestore del locale di risparmiare sui tempi e sui costi.

In particolare, i vantaggi che il gestore dell'attività otterrà dall'utilizzo di quest'app sono i seguenti:

- I camerieri potranno gestire i tavoli in maniera semplificata, facilitando la comunicazione degli ordini alla cucina.

- Il gestore del ristorante potrà organizzare i tavoli e le prenotazioni in maniera versatile e veloce.
- Sarà possibile sostituire i vecchi menù cartacei, facilitandone la consultazione da parte dei clienti e semplificandone l'aggiornamento.
- In fine il gestore avrà un modo semplice per tenersi in contatto con i propri clienti, che potranno accedere alle informazioni del ristorante direttamente dall'app.

La seguente tesi è strutturata come di seguito specificato:

- Nel Capitolo 1 sarà presentato lo sviluppo mobile multipiattaforma; partendo da una panoramica generale delle alternative disponibili per lo sviluppo mobile, verrà descritto il framework oggetto della tesi; infine sarà descritta la struttura generale delle interfacce utente di Xamarin.Forms.
- Nel Capitolo 2 verrà riportata l'analisi dei requisiti, sia quelli funzionali che quelli non funzionali.
- Nel Capitolo 3 verrà esposto il lavoro svolto durante la fase di progettazione. In particolare saranno descritte le tecnologie utilizzate per lo sviluppo dell'applicazione mobile, del database e del server, necessarie per il funzionamento del progetto.
- Nel Capitolo 4 verranno illustrati i punti salienti delle scelte compiute durante la fase di implementazione, descrivendo l'IDE utilizzato e riportando alcuni degli esempi più significativi del codice del progetto.
- Nel Capitolo 5 viene riportato il manuale utente con le schermate dell'app e il loro utilizzo.
- Nel Capitolo 6 verrà proposta una discussione in merito al lavoro svolto, in cui saranno riportate le lezioni apprese sull'utilizzo di Xamarin e una analisi SWOT del progetto.
- Nel Capitolo 7 sono riportate le conclusioni in merito al lavoro svolto e verrà proposta una beve panoramica sui possibili sviluppi futuri.

1

Sviluppo mobile Cross-Platform

In questo capitolo verrà analizzato lo sviluppo di app cross-platform. In particolare verranno elencate le principali tecnologie facendo attenzione ai vantaggi e agli svantaggi che queste comportano. Ciascuna di queste alternative coesistenti è focalizzata su contenuti, specifiche e funzionalità differenti più o meno affini alle richieste del nostro progetto.

1.1 Introduzione allo sviluppo mobile Cross-Platform

Un'applicazione mobile, o semplicemente app, è un' applicazione software progettata per essere utilizzata su periferiche mobili, come smartphone, tablet e smartwatch. Nonostante il mercato degli smartphone sia dominato principalmente da due sistemi operativi, esistono molte alternative per quanto riguarda lo sviluppo di un' app.

Le alternative principali possono essere catalogate in tre categorie:

- app native;
- progressive web app;
- cross-platform.

Esse saranno illustrate in dettaglio nelle prossime sottosezioni.

1.1.1 App native

Le app native sono app sviluppate per funzionare su un singolo sistema operativo mobile. Per ciascun sistema operativo si utilizzeranno specifici linguaggi e, quindi, specifici ambienti di sviluppo; in particolare, per sviluppare app Android si utilizza Java, o l'alternativa più recente Kotlin, e l'IDE associato sarà Android Studio. Invece il linguaggio utilizzato per programmare app per iOS è Objective-C o Swift e l'IDE associato è Xcode.

Un primo vantaggio è che esse avranno un'interfaccia nativa, quindi un aspetto e una user experience in linea con quelle del sistema operativo.

Le app native hanno, inoltre, accesso completo alle API e agli strumenti che i sistemi operativi ci fornisco; esse, pertanto sfrutteranno in maniera più efficiente

le risorse e i componenti hardware dello smartphone. Questo aspetto porta con sé prestazioni migliori e migliore stabilità delle nostre app, nonché una maggiore compatibilità con eventuali aggiornamenti del sistema operativo.

Tra gli svantaggi sicuramente va sicuramente considerato il tempo di sviluppo, in quanto saremo costretti a realizzare la stessa app due volte, adattando in linguaggi diversi e, di fatto, rendendo impossibile il riutilizzo del codice.

Riassumendo, i principali vantaggi delle app native sono i seguenti:

- UI/UX più in linea con quelle del sistema operativo;
- maggiore stabilità;
- migliori performance;
- la possibilità di sfruttare al massimo le risorse che il sistema operativo fornisce.

I principali svantaggi di questa tipologia di app sono, invece, i seguenti:

- costi elevati di sviluppo;
- maggiore tempo richiesto per lo sviluppo;
- la necessità di utilizzare tecnologie e linguaggi diversi per le due app;
- le ridotte possibilità nella scelta delle tecnologie.

1.1.2 Progressive Web App (PWA)

Lo sviluppo di Progressive Web App è un ibrido tra lo sviluppo web e lo sviluppo di app mobile native, che si pone come obiettivo quello di sfruttare gli aspetti positivi che ciascuna tecnologia può apportare ai nostri progetti. Diversamente dallo sviluppo di app native, in questo caso esistono più framework disponibili. I principali sono Angular.js, Vue.js, React.js e Ionic.

Il principale vantaggio che si ha dalla scelta di queste tecnologie è la possibilità di sviluppare un codice unico, comune per le due app, abbattendo così notevolmente i tempi e i costi necessari per lo sviluppo.

Un ulteriore vantaggio consiste nello sviluppare utilizzando i linguaggi propri dello sviluppo web, sfruttando, quindi, tecnologie molto diffuse tra gli sviluppatori, andando a contribuire all'abbattimento dei costi e dei tempi di sviluppo di nuovi progetti e dei costi di mantenimento e aggiornamento dei progetti.

D'altro canto, queste tecnologie portano con se alcuni svantaggi, infatti sfruttando questi framework difficilmente l'interfaccia utente sarà concorde con quella del sistema operativo, le performance verranno penalizzate se confrontate con quelle delle app native e l'app avrà bisogno della connessione ad internet per funzionare.

Riassumendo, i principali vantaggi delle PWA sono i seguenti:

- tempi di sviluppo ridotti;
- costi di sviluppo contenuti;
- ampia scelta di framework.

I principali svantaggi di questa tipologia di app sono, invece, i seguenti:

- UI/UX specifiche dell'app, meno in linea con il sistema operativo;
- performance ridotte;
- utilizzo ridotto delle risorse hardware e software dello smartphone;
- connessione ad Internet necessaria.

1.1.3 Cross-platform App

Lo sviluppo di app cross-platform unisce i vantaggi dei due approcci precedenti, permettendo di sviluppare app native ma riutilizzando parte del codice comune alle due app. Attraverso l'utilizzo di framework per lo sviluppo cross-platform è possibile scrivere il codice condiviso dalle nostre app Android e iOS una volta sola, esso verrà, poi, compilato per essere eseguito sullo smartphone, sfruttando le potenzialità delle risorse fornite dal sistema operativo in modo del tutto simile alle app native. Questo aspetto permette, sicuramente, di abbattere i costi e i tempi di sviluppo, pur mantenendo gran parte dei vantaggi che un'app nativa può darci, come un' interfaccia utente concorde con il sistema operativo e la possibilità di accedere a tutte le risorse fornite da quest' ultimo. Esistono più framework che permettono di sviluppare applicazioni cross-platform, i principali sono Xamarin, React Native e Flutter.

Se confrontato con lo sviluppo nativo, in quello cross-platform ci sono alcuni svantaggi, come:

- le performance leggermente inferiori ma del tutto accettabili;
- la ridotta velocità dei tempi di aggiornamento per quanto riguarda eventuali nuovi rilasci fatti da Android o iOS;
- uno sviluppo più complesso dovuto alla necessità di dover "adattare" l'app al sistema operativo specifico costringendo lo sviluppatore a lavorare anche sulla compatibilità del codice nonché ad avere delle conoscenze anche nei linguaggi nativi.

Riassumendo, i principali vantaggi delle cross-platform app sono i seguenti:

- UI/UX in linea con quelle native;
- buone performance dell' app;
- buoni tempi di sviluppo;
- bassi costi di sviluppo;
- ampia scelta di framework.

I principali svantaggi di questa tipologia di app sono, invece, i seguenti:

- Performance leggermente inferiori a quelle delle app native (trascurabili per applicazioni di piccole dimensioni);
- Sviluppo più complesso dovuto alle compatibilità;
- Necessaria conoscenza dei linguaggi nativi al fine di avere un' aspetto concorde con quello del sistema operativo;
- Tempi più lunghi per il rilascio degli aggiornamenti.

1.2 Caratteristiche di Xamarin

1.2.1 Cenni storici

Xamarin era un'azienda produttrice di software statunitense con sede a San Francisco (California), fondata nel maggio 2011 dagli ingegneri Nat Friedman e Miguel de Icaza, già noti per aver fondato circa un decennio prima l'azienda Ximian ed

aver avviato il progetto open source chiamato Mono (implementazione alternativa del compilatore C# e del framework .NET di Microsoft su sistemi Linux e macOS).

La volontà della neonata azienda era quella di offrire agli sviluppatori un modo semplice e veloce per creare app multipiattaforma attraverso nuovi e prestanti ambienti di sviluppo, basandosi su un unico linguaggio orientato agli oggetti. Da questa idea, mantenendo caratteristiche e punti forti di Mono, si è giunti alla creazione del framework Xamarin omonimo.

Negli anni il progetto ha raggiunto l'approvazione non solo degli sviluppatori (oltre 1.2 milioni), ma anche delle aziende (oltre 15 mila). Nel febbraio 2016 Microsoft ha acquisito l'azienda dando inizio a una nuova era per lo sviluppo di applicazioni multipiattaforma. A partire dal marzo 2016, inoltre, i prodotti Xamarin sono diventati ufficialmente open-source e gratuiti per sviluppatori indipendenti, piccoli team e studenti, che possono utilizzarli nei propri progetti.

Xamarin, contestualmente alla creazione del framework, realizzò un proprio ambiente di sviluppo chiamato Xamarin Studio. Questo permetteva di sviluppare le proprie applicazioni esclusivamente per Android e Windows Phone, se usato su PC Windows, oppure per Android e iOS, se usato su Mac. Oggi, a seguito dell'acquisizione di Xamarin da parte di Microsoft, Xamarin Studio non è praticamente più utilizzato. Microsoft, supportando l'utilizzo di Visual Studio, ha di fatto unificato i due IDE.

1.2.2 Struttura del framework

L'obiettivo di Xamarin è quello di fornire un framework che consenta di sviluppare app native cross-platform a partire da una stessa base di codice condivisa e utilizzando lo stesso ambiente di sviluppo. In questo modo, 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.

1.2.3 Xamarin e .NET framework

Xamarin fa parte del progetto Microsoft .NET, una piattaforma di sviluppo cross-platform e open source ideata e sviluppata da Microsoft, che mette a disposizione varie funzionalità come il supporto per più linguaggi di programmazione, modelli di programmazione asincroni e simultanei e interoperabilità, consentendo l'esecuzione su più piattaforme.

La piattaforma .NET include quindi il linguaggio C# (ed altri come F#), gli strumenti per la compilazione, le librerie di base per lo sviluppo, gli editor e strumento per lo sviluppo per Windows, Linux e MacOS (Figura 1.1) [5].

1.2.4 Xamarin Native e Xamarin.Forms

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

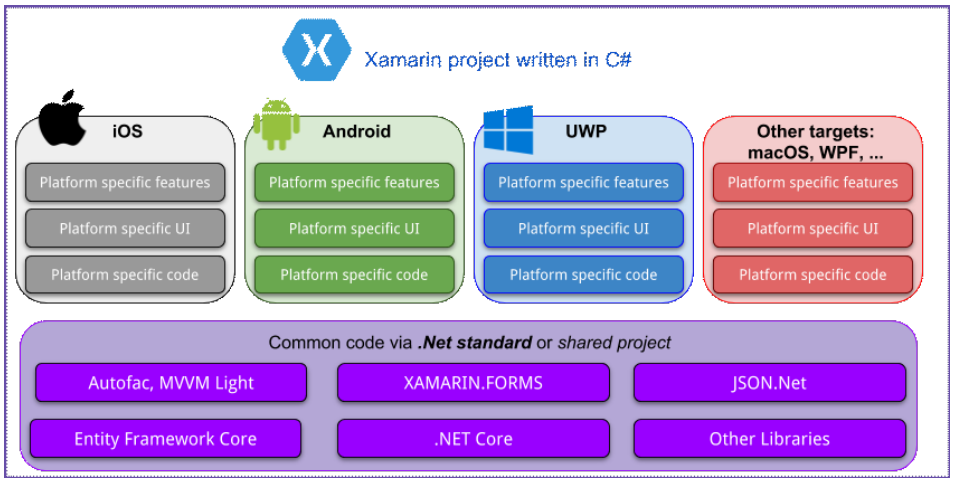

Figura 1.1. Xamarin è un progetto appartenente all'ecosistema .Net

"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 è possibile richiamare qualsiasi metodo esposto dall'omonimo SDK. Allo stesso modo, Xamarin.iOS fornisce agli sviluppatori l'accesso all'SDK completo di iOS (Figura 1.2).

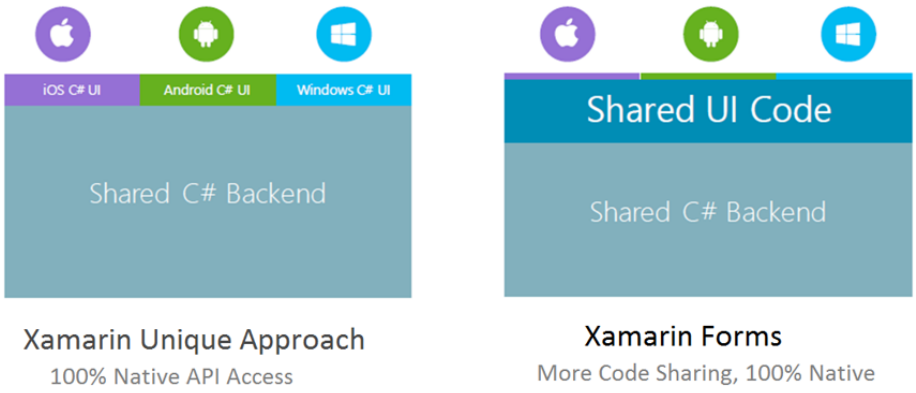

Figura 1.2. Xamarin permette di sviluppare app con approccio Native o sfruttando Xamarin.Forms

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). Nel seguito, analizzeremo proprio quest'ultimo approccio.

1.3 Alternative di code sharing

Un progetto Xamrin relativo ad un'app può essere suddiviso in due sotto-progetti, ovvero:

- *Core Project*: è essenzialmente il cuore della nostra applicazione; esso contiene gli elementi essenziali, quali l'accesso ai dati e a sevizi remoti nonché la logica di business del nostro progetto.
- *Platform-Specific Application Projects*: è composto dai progetti delle singole piattaforme supportate dal nostro progetto. In particolare, ogni progetto dovrebbe essere costituito dalla gestione dell'interfaccia utente (UI) e dalla gestione di specifiche funzionalità delle piattaforme tra il layer relativo alla logica di business della nostra applicazione e l'UI.

Il Core Project contiene, quindi, il codice condiviso dai nostri progetti per le specifiche piattaforme; è importante però analizzare le possibili modalità di condivisione di tale codice. Innanzitutto, il core project si basa sul pattern Layers, che consiste, sostanzialmente, sulla suddivisone del codice in insiemi di strati, tra questi citiamo:

- Il *Business Layer*, che conterrà tutta la logica e le operazioni necessarie per l'interazione con l'utente.
- Il *Data Access Layer*, che fornisce al Business Layer, attraverso opportune interfacce, l'accesso ai dati del database.

Due sono, quindi, le principali modalità di condivisione del codice comune tra i diversi progetti delle specifiche piattaforme supportate dalla nostra soluzione: Shared Assets Project (SAP) e Portable Class Libraries (PCLs).

1.3.1 Shared Assets Project (SAP)

Sfruttando le direttive al compilatore, esso permette di includere differenti porzioni di codice, specifiche per ogni piattaforma, condividendo, quindi, il codice attraverso l'intera soluzione (Figura 1.3).

Sostanzialmente, lo Shared Assets (e i suoi Layer) vengono copiati in ogni progetto della soluzione che lo referenzia, e viene compilato come parte del progetto. Ciò avviene attraverso delle direttive al compilatore in C# di tipo condizionale (Listato: 1.1):

```
#if
#elif
#endif
```

Listing 1.1. Istruzioni condizionali utilizzabili per il compilatore

A tali istruzioni si affiancano opportuni simboli utili ad identificare univocamente il codice specifico per la singola piattaforma supportata dal nostro progetto, come mostrato nella Tabella 1.1:

Tramite la combinazione dei simboli e delle condizioni `#if` ed `#elif`, possiamo integrare funzionalità specifiche delle piattaforme di interesse aggiungendo, nei

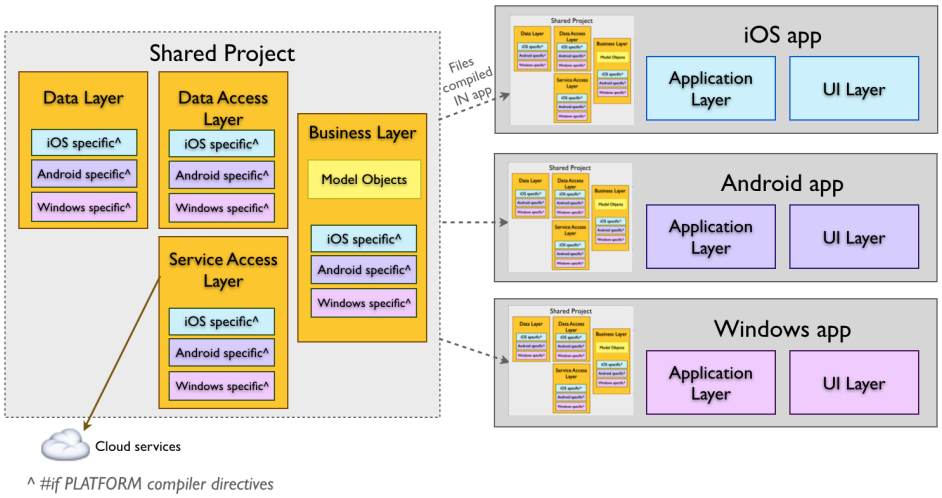

Figura 1.3. Shared Assets: questo approccio fa sì che la condivisione del codice tra i progetti avvenga nella fase di compilazione della soluzione

Simbolo	Descrizione
`__MOBILE__`	Definito nei progetti mobile
`__IOS__`	Definito nel progetto iOS
`__ANDROID__`	Definito nel progetto Android
`__ANDROID_nn__`	Alternativa per la definizione di blocchi di codice Android, con la possibilità di identificare il livello delle API (indicando il numero corrispondente al posto di "nn")
`WINDOWS_PHONE_APP`	Definito nel progetto per Windows Phone (WP)
`WINDOWS_UWP`	Utilizzato nel progetto delle Universal Windows Platform (UWP)

Tabella 1.1. SAP: direttive al calcolatore

metodi che lo richiedono, un blocco di istruzioni condizionali secondo le modalità mostrate nel Listato 1.2.

```
#if __IOS__
        // codice specifico per iOS
        #elif __ANDROID__
        // codice specifico per Android
        #elif WINDOWS_PHONE_APP
        // codice specifico per Windows Phone
        ... //supporto agli altri progetti
        #endif
```

Listing 1.2. Esempio di utilizzo delle direttive al compilatore

In questo modo il codice sarà compilato per ogni progetto ma i blocchi di codice identificati opportunamente saranno compilati solo per la specifica piattaforma [1].

1.3.2 Portable Class Libraries (PCLs)

L'approccio alternativo a quello SAP si propone di gestire il codice condiviso tra i vari progetti delle specifiche piattaforme attraverso l'utilizzo di un sottoinsieme di librerie offerte dal .NET Framework, sfruttando, quindi, il design pattern `Dependency Injection` (DI) (Figura 1.4).

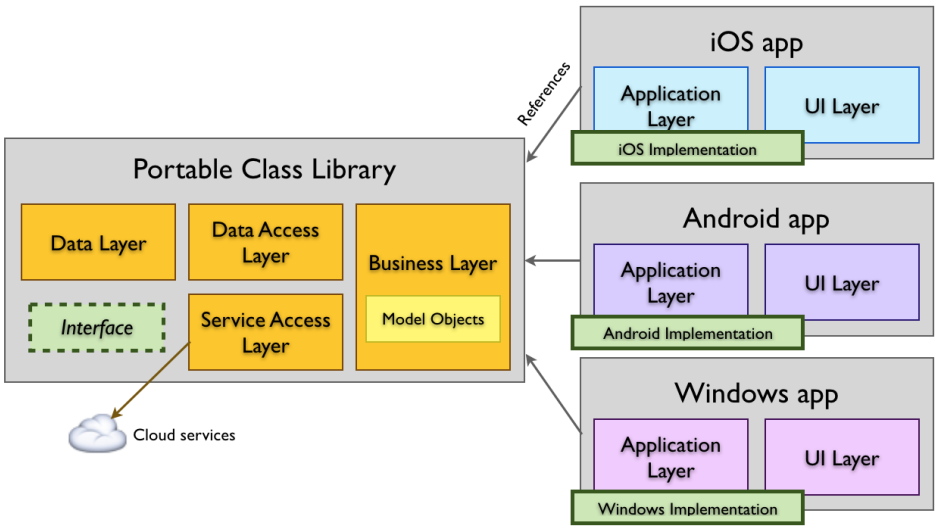

Figura 1.4. Portable Class Libraries (PCLs): tale approccio permette di usare un sottoinsieme di librerie offerte dal .NET framework per sfruttare lo stesso codice nei progetti che compongono l'intera soluzione

Sostanzialmente, per applicare il DI, nel codice condiviso saranno definite le interfacce che poi verranno implementate in ogni piattaforma tramite sottoclassi. Xamarin fornisce la classe `DependencyService`, necessaria ad implementare il pattern DI. Per sfruttare questa classe verrà definita un'interfaccia `INomeClasse`, definita nel PCL invocando il metodo `Get` di `DependencyService` (Listato 1.3).

```
DependencyService.Get<INOMECLASSE>.Nome_Metodo
```

Listing 1.3. Metodo `Get` di `DependencyService`

All'interno del progetto della specifica piattaforma, verrà implementata attraverso il tag [`assembly`] come mostrato nel Listato 1.4.

```
[assembly: Xamarin.Forms.Dependency(typeof(NomeClasse_PIATTAFORMA))]
```

Listing 1.4. chiamata del tag `assembly` di `DependencyService`

dove `NOMECLASSE_PIATTAFORMA` è il nome della classe relativo alla piattaforma (Android, iOS, Windows Phone, etc.) che la implementa.

1.3.3 Confronto tra SAP e PCLs

Entrambi gli approcci mostrano dei vantaggi e degli svantaggi che andranno valutati in relazione al progetto specifico.

L'approccio PCLs verrà utilizzato esclusivamente nel caso in cui si dovrà ricorrere a funzionalità specifiche di una piattaforma, evitando di dover includere

direttive al compilatore. Il refactoring del codice sarà più semplice con l'approccio PCL rendendo potenzialmente più semplice il mantenimento della soluzione [1].

1.4 UI in Xamarin.Forms

Lo scopo di Xamarin.Forms è quello di avere una codebase condivisa tra le piattaforme del nostro progetto, non solo per quanto riguarda la logica dell'app ma anche l'interfaccia utente (UI).

Xamarin.Forms, infatti, fornisce una serie di classi per la definizione dell'interfaccia attraverso l'utilizzo di file XAML (Extensible Application Markup Language) o di codice in C#.

Sono tre le principali classi che compongono la UI:

- `Page`;
- `Layout`;
- `View`.

La classe che sta alla base di ogni schermata è di tipo `Page` essa conterrà quindi al suo interno tutte le classi di tipo `Layout` della schermata stessa. La classe `Layout` potrà contenere a sua volta altre classi di tipo `Layout` o cassi di tipo `View` (Figura 1.4).
Queste ultime saranno i veri e propri elementi di interazione tra UI e l'utente stesso.

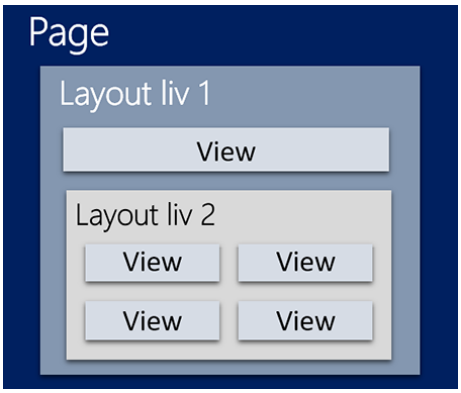

Figura 1.5. Schema degli elementi che compongono l'interfaccia su Xamarin.Forms

1.4.1 Page

La `Page` di Xamarin.Forms rappresenta una schermata della nostra app e quindi corrisponde ad un View Controller in iOS, ad una Activity in Android oppure ad una Page in WP/UWP.

Xamarin.Forms fornisce varie tipologie di `Page` per visualizzare gli elementi della UI (Figura 1.6).

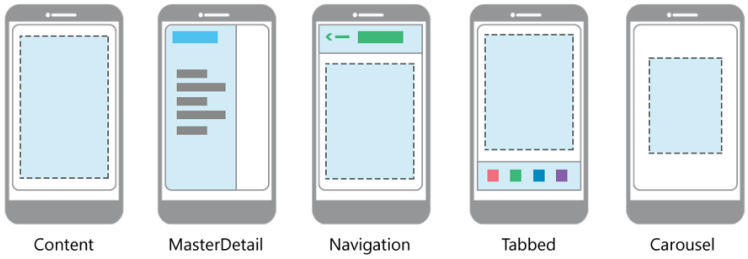

Figura 1.6. Le principali tipologie di `Page` fornite da Xamarin.Forms. Tali tipologie vengono descritte nella Tabella 1.2

Page	Descrizione
`ContentPage`	Consiste in una singola schermata nella quale verrà definito un contenitore di tipo Layout in cui a sua volta sarà definiti un elemento di tipo View
`MasterDetailPage`	Offre la possibilità di gestire due pannelli (Pane). Il `Master Pane` contiene un elenco di risorse a cui è associato un Detail Pane che mostra informazioni in dettaglio
`NavigationPage`	Gestisce la navigazione e lo stack delle pagine che l'utente può visualizzare
`TabbedPage`	È una pagina che contiene sulla parte superiore dello schermo una serie di Tab che richiamano una specifica schermata
`CarouselPage`	Permette di definire una serie di pagine di tipo `ContentPage` attraverso le quali si può navigare scorrendo il dito (swipe)

Tabella 1.2. Pages: tipi e descrizioni

1.4.2 Layout

Come già anticipato, i Layout sono dei contenitori al cui interno è possibile definire altri elementi di tipo `Layout` o elementi di tipo `View`. Queste classi possono essere suddivise in due gruppi:

Layout con un solo elemento figlio

L'aspetto di questa tipologia di layout è riportata in Figura 1.7

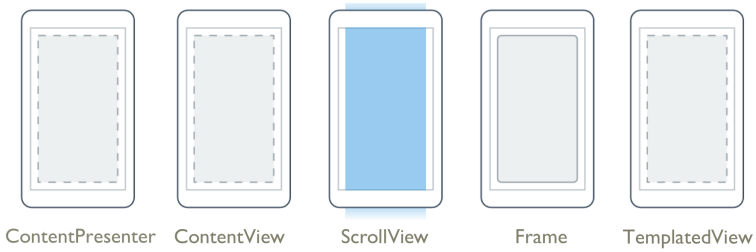

Figura 1.7. Aspetto dei layout con un solo elemento figlio

Nella Tabella 1.3 vengono descritti tali `Layout`.

Tipologia	Descrizione
`ContentPresenter`	E' un gestore di layout per le visualizzazioni basate su modelli, utilizzato all'interno di un oggetto `ControlTemplate` per contrassegnare il punto in cui viene visualizzato il contenuto
`ContentView`	Viene usato principalmente come elemento strutturale e funge da classe di base per `Frame`. Del Content è possibile definire il Background e il Padding. Estende la classe `TemplatedView`
`Frame`	Letteralmente "cornice", fornisce la possibilità di definire l'effetto ombra o il colore del bordo
`ScrollView`	È utilizzato per creare pagine scorrevoli. Permette di definire la direzione dello scroll e al proprio interno ulteriori contenuti
`TemplatedView`	È una `View` che mostra un contenuto tramite un `ControlTemplate`; è impiegata come classe di base per la `ContentView`

Tabella 1.3. Layout con un solo elemento figlio: tipi e descrizioni

Layout con più elementi figli

L'aspetto di queste tipologia di `Layout` è riportato in Figura 1.8.

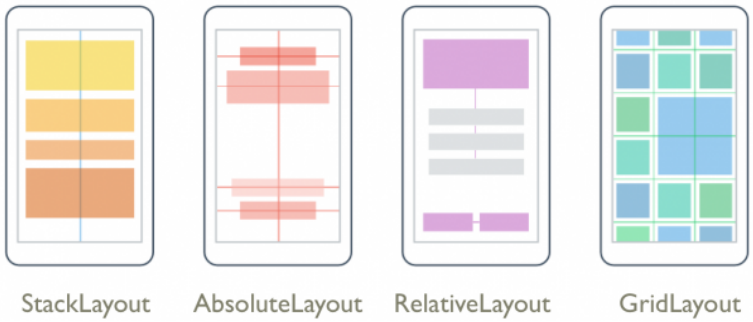

Figura 1.8. Aspetto dei layout con un più elementi figli

Nella Tabella 1.4 vengono descritti tali `Layout`.

Tipologia	Descrizione
StackLayout	Gli elementi figli contenuti in questo layout possono essere disposti verticalmente o orizzontalmente
AbsoluteLayout	Imposta la posizione e le dimensioni degli elementi figli utilizzando dei rettangoli o definendo delle specifiche proporzioni
RelativeLayout	Permette di definire dei vincoli tra gli elementi figli e per poter definire la posizione e la dimensione al suo interno. Tra tutti, è il layout che offre una maggiore libertà nella disposizione degli elementi e un layout più fluido
Grid	Crea un contenitore organizzato secondo una griglia con un certo numero di righe, colonne e larghezza. Ogni cella permette di gestire al suo interno altri elementi di tipo View

Tabella 1.4. Layout con più elementi figli: tipi e descrizioni

1.4.3 View

Le `View` sono gli elementi fondamentali delle UI sviluppate con Xamarin, in quanto sono gli elementi di interazione con cui l'utente si interfaccerà [6].

I principali esempi di elementi di tipo View sono i seguenti:

- *BoxView*: visualizza un rettangolo a tinta unita colorato.
- *Label*: visualizza le stringhe di testo a riga singola o i blocchi di testo a più righe (Figura 1.9).

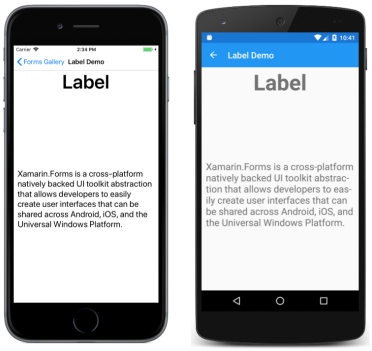

Figura 1.9. Esempio di Label

- *Expander*: fornisce un contenitore espandibile per ospitare qualsiasi contenuto; è costituito da un'intestazione e da un contenuto (Figura 1.10).

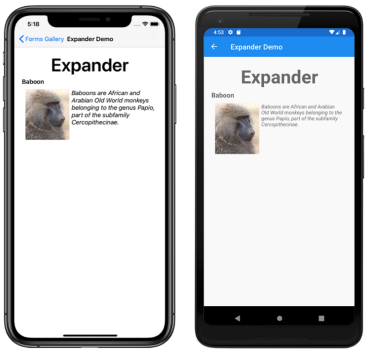

Figura 1.10. Esempio di Expander

- *Image*: visualizza un'immagine.
- *Map*: visualizza una mappa, sfruttando il pacchetto NuGet (Figura 1.11).
- *Media element*: riproduce video o audio.
- *Button*: elemento rettangolare che visualizza un testo e genera un evento Clicked quando viene premuto (Figura 1.12).

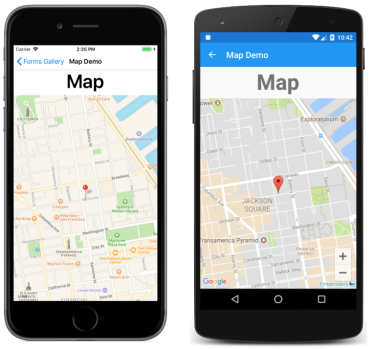

Figura 1.11. Esempio di Map

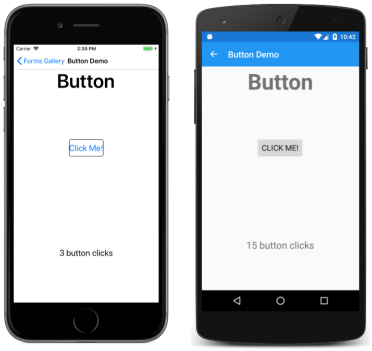

Figura 1.12. Esempio di Button

- *ImageButton*: visualizza un'immagine cliccabile.
- *RadioButton*: consente la selezione di un'opzione (Figura 1.13).

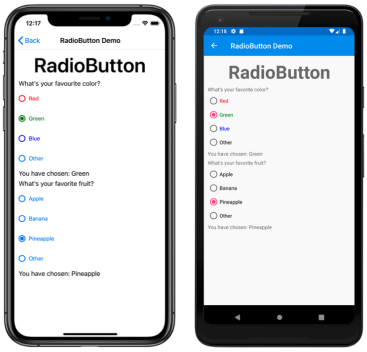

Figura 1.13. Esempio di RadioButton

- *SearchBar*: visualizza una barra di ricerca (Figura 1.14).

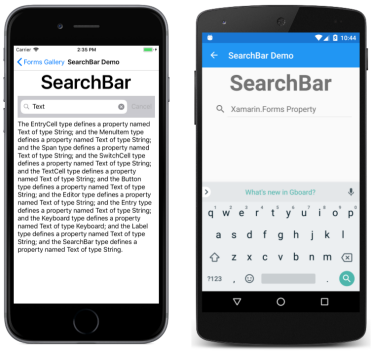

Figura 1.14. Esempio di SearchBar

- *CheckBox*: visualizza una casella restituendo un valore booleano. *Slider*: Consente all'utente di selezionare un valore in un intervallo continuo;
- *Switch*: elemento che assume la forma di un valore di attivazione/disattivazione per consentire all'utente di selezionare un valore booleano (Figura 1.15).

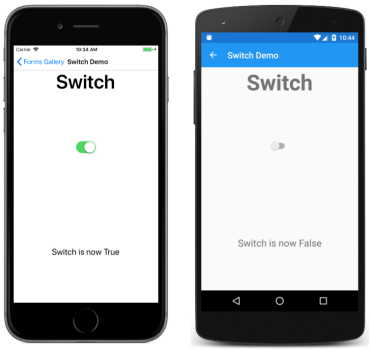

Figura 1.15. Esempio di Switch

- *DatePicker*: consente all'utente di selezionare una data (Figura 1.16).
- *TimePicker*: consente all'utente di selezionare un orario (Figura 1.17).
- *Entry*: consente all'utente di immettere e modificare una singola riga di testo (Figura 1.18).
- *CollectionView*: consente di visualizzare un elenco scorrevole di elementi di dati selezionabili (Figura 1.19);
- *ListView*: consente di visualizzare un elenco scorrevole di elementi di dati selezionabili.
- *TableView*: consente di visualizzare una tabella (Figura 1.20).

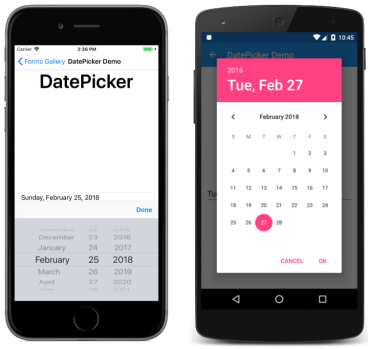

Figura 1.16. Esempio di DatePicker

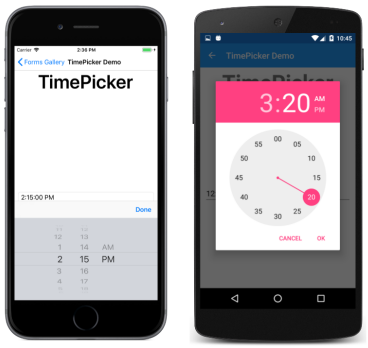

Figura 1.17. Esempio di TimePicker

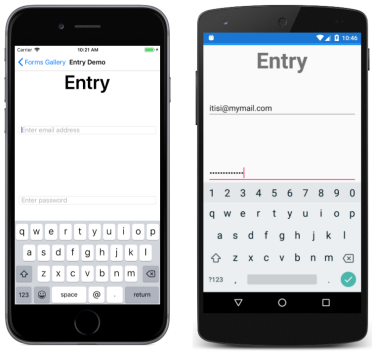

Figura 1.18. Esempio di Entry

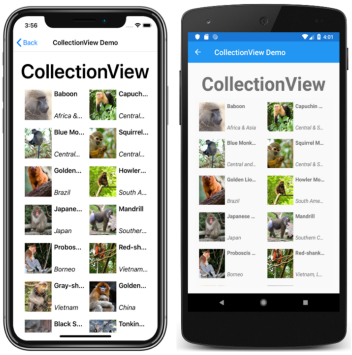

Figura 1.19. Esempio di CollectionView

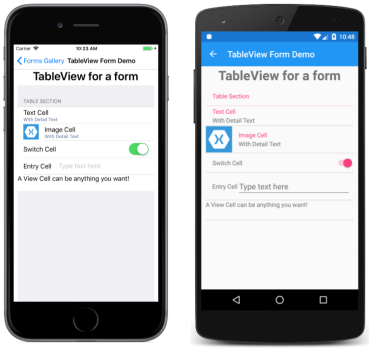

Figura 1.20. Esempio di TableView

2

Analisi dei requisiti

In questo capitolo vengono trattate la descrizione generale e l'analisi dei requisiti dell'app oggetto della presente tesi.

2.1 Descrizione dell'app

Il progetto consiste nell'implementazione di un'app pensata per la gestione degli ordini di un ristorante. L'app potrà essere installata sia dal cliente che dal personale del ristorante con l'obiettivo di rendere facilmente accessibile il menù e velocizzare la gestione delle ordinazioni del ristorante.

Le principali funzionalità che la nostra app fornirà sono le seguenti:

- *Il cliente potrà:*
  - prenotare un tavolo;
  - visualizzare il menù;
  - fare un ordine;
  - scegliere le modalità di pagamento.
- *Il cameriere potrà:*
  - assegnare un tavolo;
  - gestire le ordinazioni;
  - visualizzare lo stato di preparazione degli ordini.
- *Il cuoco potrà:*
  - visualizzare le ordinazioni;
  - inviare una notifica al cameriere se il piatto è pronto.

L'app permette di ridurre i tempi di attesa, consentendo al cliente di visualizzare il menù direttamente dallo schermo del proprio telefono. Il personale, quindi, non dovendo più fornire il menù ai clienti, potrà rimanere a disposizione di essi per chiarimenti, effettuando l'ordine con più facilità. Inoltre, vi è più flessibilità per ogni cliente nel pagamento e nelle prenotazioni.

Il proprietario del locale non dovrà più gestire le ordinazioni, né fare il conto di ogni tavolo, perché è già tutto integrato nell'app.

2.2 Analisi dei requisiti

In questa sezione viene proposta un'analisi dei requisiti funzionali e non funzionali del progetto.

I requisiti funzionali rappresentano l'insieme delle caratteristiche che dovranno essere implementate.

Invece, i requisiti non funzionali indicano l'insieme dei vincoli realizzativi che l'applicazione dovrà rispettare.

2.2.1 Requisiti funzionali

I requisiti funzionali che la nostra app dovrà rispettare sono di seguito elencati:

- l'app dovrà fornire all'amministratore, il pieno controllo;
- l'app dovrà gestire le funzionalità CRUD (Create, Read, Update e Delete) del titolare/amministratore;
- l'app dovrà gestire le funzionalità CRUD dei clienti;
- l'app dovrà gestire le funzionalità CRUD dei dipendenti;
- l'app dovrà consentire all'amministratore di creare, modificare o rimuovere l'account di un dipendente;
- l'app dovrà consentire all'amministratore di visualizzare le prenotazioni;
- l'app dovrà consentire all'amministratore di gestire gli ordini;
- l'app dovrà consentire all'amministratore di modificare il menù;
- l'app dovrà consentire all'amministratore di gestire i pagamenti;
- l'app dovrà consentire all'amministratore di assegnare i tavoli ai camerieri;
- l'app dovrà consentire al cliente di registrarsi in modo tale da avere un account;
- l'app dovrà consentire al cliente di prenotare un tavolo;
- l'app dovrà consentire al cliente di visualizzare il menù;
- l'app dovrà consentire al cliente di fare un ordine;
- l'app dovrà consentire al cliente di scegliere la modalità di pagamento;
- l'app dovrà consentire al cuoco di visualizzare l'ordine;
- l'app dovrà consentire al cuoco di inviare una notifica al cameriere;
- l'app dovrà consentire al cameriere di visualizzare a quali tavoli è stato assegnato e le notifiche da parte del cuoco.

2.2.2 Requisiti non funzionali

I requisiti non funzionali rappresentano i vincoli e le proprietà/caratteristiche relative a un sistema, come vincoli di natura temporale, vincoli sul processo di sviluppo e sugli standard da adottare. Tali requisiti non riguardano solo il sistema software che si sta sviluppando; alcuni, infatti, possono vincolare il processo usato per realizzare il sistema

I requisiti non funzionali che la nostra app dovrà rispettare sono:

- Al momento della registrazione di un nuovo profilo l'app dovrà richiedere ai clienti uno username di almeno 6 caratteri, che dovrà essere diverso da quello degli altri utenti, e una password di almeno 6 caratteri, oltre a vari dati personali.

- Il linguaggio di programmazione utilizzato per sviluppare l'app è C#.
- Verrà sviluppata una piattaforma di backend in PHP, su una macchina LAMPP.
- L'IDE utilizzato per la stesura del codice dell'app è Visual Studio.
- Verranno utilizzate varie librerie e API per semplificare la stesura del codice.
- Il database sarà implementato utilizzando MySQL.
- PhpMyAdmin sarà l'applicazione web che permetterà l'amministrazione di MySQL.
- Il software utilizzato per testare le API REST è Postman.
- Il software per la creazione dei mockup è Balsamiq Mockups 3.

3

Progettazione

In questo capitolo analizzeremo l'app più da vicino. In particolare, verranno descritti gli aspetti principali della progettazione, ponendo particolare attenzione alle tecnologie utilizzate per lo sviluppo dell'app.

3.1 Descrizione dell'app

La fase di progettazione si occupa di definire come il sistema dovrà rendere possibili le operazioni discusse nel processo di analisi dei requisiti.

Di fatto, può essere vista come la creazione di una soluzione software ad un problema. Proprio per questo durante la progettazione bisogna tenere in considerazione i fattori, le potenzialità e i limiti relativi alle tecnologie che verranno adoperate per la realizzazione del progetto.

Per fare ciò si è pensato di sviluppare un' app mobile che si interfaccerà con un database remoto attraverso delle API REST, in modo da rendere il servizio accessibile sia da diverse periferiche mobili che da desktop via web, come schematizzato nella Figura 3.1.

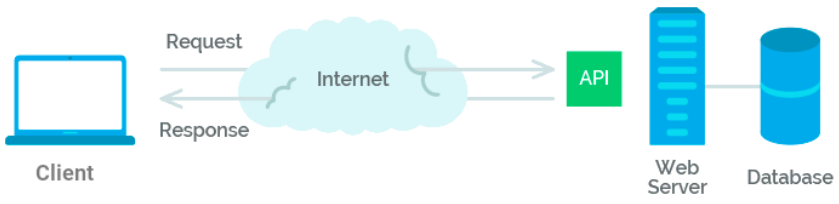

Figura 3.1. Schematizzazione della comunicazione tra client e server in un'architettura REST

Nelle prossime sezioni verrà descritta, in maggiore dettaglio, la struttura del nostro progetto, che si compone di tre principali elementi:

- il client (la nostra app mobile);
- il database;
- le API REST.

3.2 Applicazione mobile

Per la realizzazione dell'app mobile è stato utilizzato Xamarin (Figura 3.2), framework in C#.

Figura 3.2. Logo di Xamarin

La nostra app si interfaccerà, come già anticipato, con un database attraverso delle API descritte nella prossima sezione.

3.2.1 Mockup

In fase di progettazione sono stati creati dei mockup utili al fine di stabilire una struttura chiara delle schermate dell'app; per la realizzazione di questi mockup è stato utilizzato Balsamiq mockup 3.

Di seguito alcuni esempi:

- la struttura generale dell'app come una Master Detail, viene riportata nella Figura 3.3;
- la schermata con cui l'utente potrà aggiungere un prodotto al proprio ordine, viene mostrata nella Figura 3.4;
- la schermata con cui il cliente potrà interagire con il conto, viene riportata nella Figura 3.5;
- la schermata con cui il cliente potrà prenotare un tavolo, viene illustrata nella Figura 3.6.

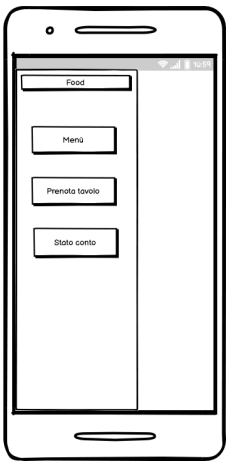

Figura 3.3. Struttura del Master Detail

Figura 3.4. Aggiunta di prodotti da parte del cliente

3.2.2 Visual Studio

Dopo l'acquisizione da parte di Microsoft, Xamarin è diventato ufficialmente open source e gratuito. Microsoft, infatti, offre pieno accesso al framework e il permesso di utilizzo in tutte le versioni, precedentemente fornite sotto licenza a pagamento.

L'IDE, utilizzato per lo sviluppo dell'app mobile, è Visual Studio (Figura 3.7). La scelta è ricaduta su di esso in quanto, oltre ad essere l'IDE di riferimento per Xamarin, la versione community è rilasciata gratuitamente per fini didattici.

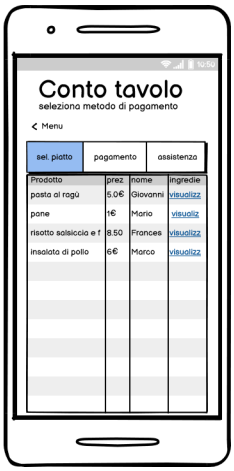

Figura 3.5. Visualizzazione del conto da parte del cliente

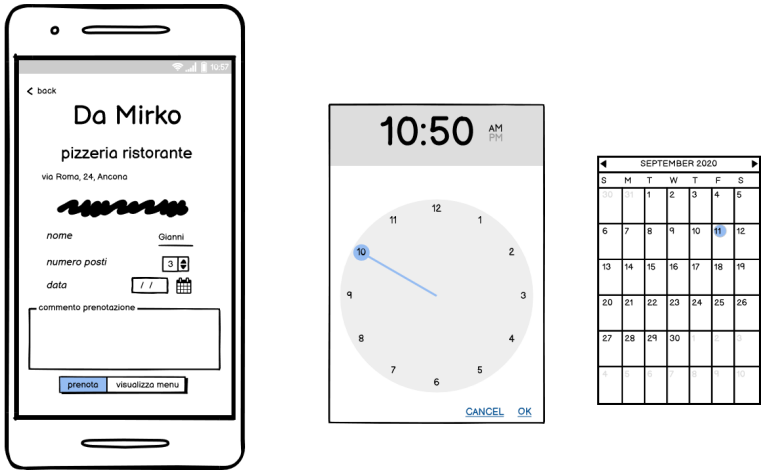

Figura 3.6. Prenotazione del tavolo da parte del cliente

Figura 3.7. Logo di Visual Studio

3.3 Database

Il database, utile per la memorizzazione dei dati necessari, verrà realizzato sfruttando l'RDBMS (Relational Database Management System) MySQL (Figura 3.8).

MySQL è RDBMS open source sin dalla nascita; esso rappresenta una delle tecnologie più note e diffuse del settore.

Figura 3.8. Logo di MySQL

3.3.1 PhpMyAdmin

Il nostro database MySQL viene gestito con PhpMyAdmin, il quale fornisce un'interfaccia grafica per la gestione e la visualizzazione delle tabelle, che lo compongono.

3.4 Server back-end

Al fine di interagire con il database è stato necessario realizzare delle API che implementassero le operazioni CRUD (Create, Read, Update, Delete) associate ad ogni risorsa.

Lo sviluppo delle API è avvenuta attraverso il framework chiamato Laravel (Figura 3.9), seguendo lo stile architetturale REST (Representational State Transfer).

REST non è una tecnologia specifica ma, piuttosto, un insieme di linee guida per la realizzazione di una "architettura di sistema" e una metodologia di progettazione. Essa descrive un'applicazione in termini di risorse e di operazioni standard. Le risorse sono il concetto fondamentale di questa architettura; ogni risorsa è associata ad un identificatore globale (URI) e rappresenta le entità del mondo reale.

Le operazioni standard, che vengono mappate sui metodi HTTP (GET, POST, PUT, DELETE), rappresentano le opzioni disponibili per interagire con le risorse; ad ogni richiesta si riceveranno i dati in risposta nel formato JSON.

Figura 3.9. Logo di Laravel

3.4.1 PhpStorm

La stesura del codice PHP è avvenuta utilizzando l'ambiente di sviluppo PhpStorm (Figura: 3.10), sicuramente uno degli IDE più completi e versatili disponibili, realizzato dalla società JetBrains.

Figura 3.10. Logo di PhpStorm

3.4.2 Deploying

Il deploy del back-end del nostro progetto avverrà su piattaforma LAMP, uno dei più noti e diffusi stack per applicazioni web. Il termine LAMP deriva dalle iniziali delle componenti software su cui lo stack si basa:

- L - Linux, il sistema operativo (nel nostro caso Ubuntu);
- A - Apache, il server web;
- M - MySQL, l'RDBMS (Relational Database Management System);
- P - PHP, il linguaggio di scripting utilizzato per sviluppare applicazioni web.

Nel nostro progetto DigitalOcean sarà la piattaforma cloud che farà da host del back-end.

3.5 Workflow

La Figura 3.11 schematizza chiaramente il flusso di lavoro dell'intero progetto; attraverso delle opportune chiamate, definite dalle API, sarà possibile accedere alle risorse necessarie alla nostra app, ricevendo risposte in formato JSON.

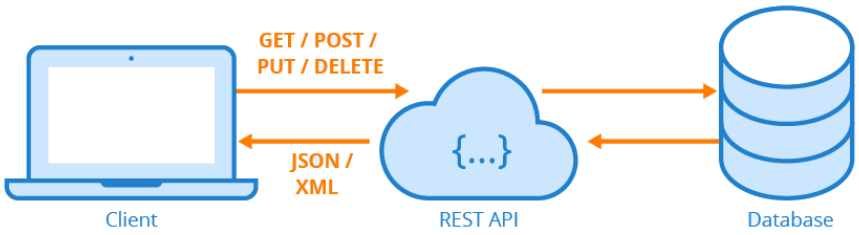

Figura 3.11. Schematizzazione della struttura dell'intero progetto

4

Implementazione

In questo capitolo illustreremo l'implementazione dell'app oggetto della presente tesi.

4.1 Visual Studio

Xamarin è una piattaforma di sviluppo .NET composta da strumenti, linguaggi di programmazione e librerie per la creazione di svariati tipi di applicazioni.

Xamarin fornisce componenti aggiuntive, che consentono agli sviluppatori di creare app Android, iOS e Windows Phone, sfruttando allo stesso tempo gli strumenti forniti da Visual Studio. L'IDE, infatti, fornisce la possibilità di sfruttare alcune tecnologie, come il completamento del codice e IntelliSense, che permettono di velocizzare e facilitare il lavoro del programmatore durante la fase di sviluppo del codice.

Xamarin per Visual Studio dispone anche di estensioni che forniscono supporto per la creazione, la distribuzione e il debug di app, consentendo allo sviluppatore di testare le funzionalità dell'app utilizzando un simulatore o un dispositivo fisico esterno.

Dopo l'avvio della collaborazione tra Xamarin e Microsoft sono state introdotte ulteriori tecnologie tra le quali il supporto per le librerie di Microsoft e la maggior parte delle funzionalità di C#, come `async/await`, per le chiamate asincrone.

4.1.1 Installazione e configurazione dell'ambiente di sviluppo

Accedendo al sito di Microsoft Visual Studio (https://visualstudio.microsoft.com) è possibile scaricare l'IDE, nel nostro caso nella versione Community Edition.

Dopo aver scaricato ed avviato il file eseguibile *Vs_ Community.exe*, dovremo accettare le condizioni della licenza di utilizzo. Il programma di installazione, quindi, avvierà un processo, al termine del quale il nostro computer sarà pronto per l'istallazione dell'IDE.

Successivamente sarà possibile selezionare l'ambiente di sviluppo; nel nostro caso andrà selezionata la spunta, associata all'opzione "Mobile development with .NET", come illustrato nella Figura 4.1.

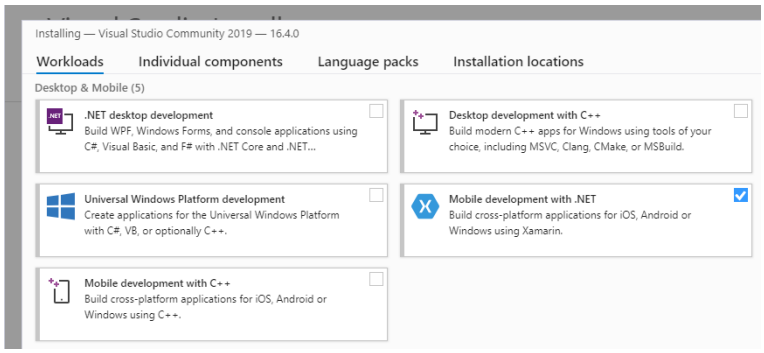

Figura 4.1. In questa schermata è possibile selezionare l'ambiente di sviluppo di nostro interesse

Dopo aver selezionato l'opzione corretta, il wizard si occuperà di selezionare i pacchetti necessari, altrimenti selezionabili singolarmente. Si avvierà, quindi, un processo, al termine del quale, il nostro IDE sarà installato e configurato correttamente, pronto per permetterci di sviluppare codice.

Aprendo Visual Studio, sarà possibile creare un nuovo progetto selezionando nel menù, alla voce *File|Nuovo|Progetto*, oppure effettuando l'analoga versione della pagina iniziale all'avvio dell'IDE.

Nella finestra *Nuovo Progetto*, riportata nella Figura 4.2, possiamo selezionare la nostra soluzione cross-platform.

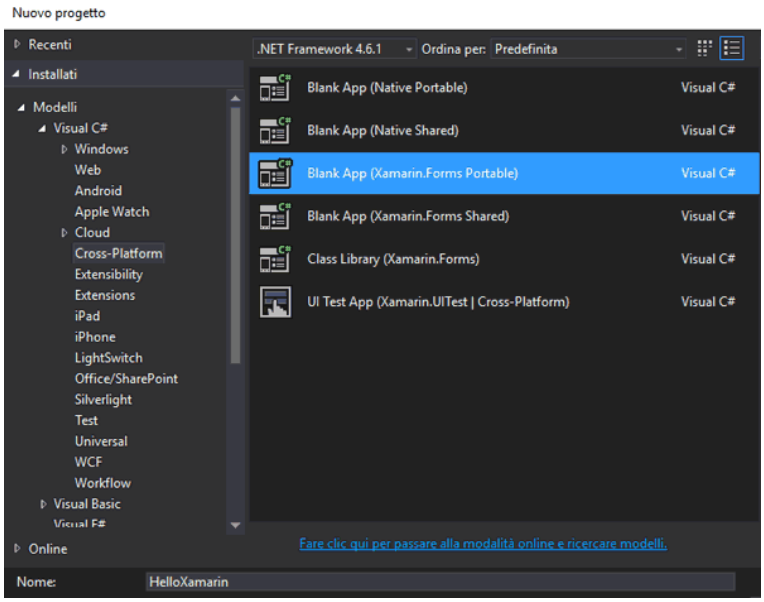

Figura 4.2. In questa schermata ci sono alcune delle alternative selezionabili per creare un nuovo progetto

Selezionando la sezione di nostro interesse (*Modelli\ Visual C#\ Cross-Platform*) visualizzeremo le soluzioni di partenza per la nostra app, brevemente descritte nella Tabella 4.1.

Tipo di app		Descrizione
Blank App	Native Portable	Soluzione per la creazione di app native cross-platform condividendo il codice tramite Portable Class Library (PCL)
	Native Shared	Crea una soluzione per la realizzazione di app native cross-platform condividendo il codice tramite uno Shared Assets Project
	Xamarin.Forms Portable	Soluzione per la creazione di app cross-platform con Xamarin.Forms condividendo il codice tramite PLC
	Xamarin.Forms Shared	Crea una soluzione per la realizzazione di app cross-platform con Xamarin.Forms condividendo il codice tramite Shared Assets Project
Class Library	Xamarin.Forms	Permette di creare una class library che può essere referenziata in qualsiasi progetto Xamarin.Forms
UI Test App	Xamarin.UITest / Cross-Platform	Crea un progetto utilizzando le feature di test automatizzate della User Interface (UI) di Xamarin)

Tabella 4.1. Elenco sintetico delle possibili soluzioni che possiamo utilizzare per la creazione di un nuovo progetto

4.1.2 Inizializzazione di una nuova soluzione

Scegliendo *Blank App (Xamarin.Forms Portable)* inizializzeremo una nuova soluzione cross-platform, del tutto priva di ogni struttura predefinita.

La nostra soluzione "HelloXamarin" sarà divisa in 6 progetti, come mostrato nella Figura 4.3.

Ogni progetto contiene il codice di una piattaforma specifica; i particolare:

- nel progetto "HelloXamarin (Portatile)" scriveremo il codice condiviso da tutta la soluzione, come la logica di business e la User Interface dell'app mancante di eventuali adattamenti.
- Nel progetto "HelloXamarin.Droid" inseriremo il codice nativo per Android.
- Nel progetto "HelloXamarin.iOS" inseriremo il codice nativo per iOS.
- Nel progetto "HelloXamarin.UWP" inseriremo il codice nativo per UWP (Universal Windows Platform).
- Nel progetto "HelloXamarin.Windows" inseriremo il codice nativo per le app desktop Windows.
- Nel progetto "HelloXamarin.WinPhone" inseriremo il codice nativo per Windows Phone; questo è, di fatto, un sistema operativo caduto in disuso; Windows, infatti, non fornisce più supporto per questo sistema operativo.

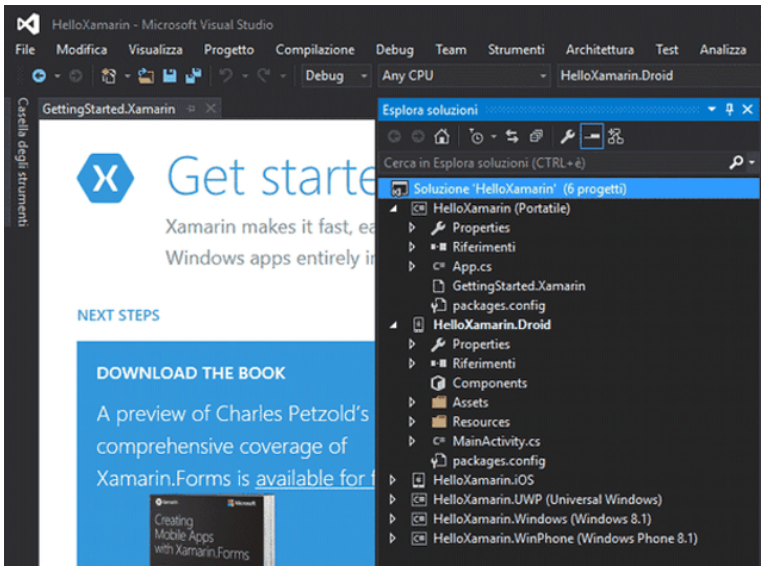

Figura 4.3. La soluzione sarà divisa in 6 progetti corrispondenti alle differenti piattaforme

All'interno del progetto HelloXamarin troviamo il file *App.cs*, esso implementa una classe che definisce la pagina iniziale, poi referenziata in tutti i progetti specifici delle varie piattaforme.

Le rispettive classi create in automatico negli altri progetti sono: *MainActivity.cs*, per il progetto Android, *AppDelegate.cs* per il progetto iOS, e *MainPage.xaml.cs* per UWP, Windows e Windows Phone. Queste consistono nella prima pagina visibile all'avvio della nostra app, inizialmente definita come una pagina vuota.

4.1.3 Il ciclo di vita di un'app in Xamarin

La classe `Application` rappresenta la base di un'app cross-platform creata con Xamarin.Forms, e si compone delle seguenti proprietà:

- La proprietà *Current* è statica e contiene un riferimento all'oggetto dell'applicazione.
- La proprietà *MainPage* definisce la prima pagina della nostra app.
- La proprietà *Properties* non è altro che un dizionario, che estende l'interfaccia *IDictionary<string, object>*, ed è utile per memorizzare dati da mantenere durante i cambi di stato dell'app.
- Anche la proprietà *Resources* è un dizionario, in questo caso di tipo *ResourceDictionary*, composto da coppie chiave-valore, dove le chiavi sono di tipo *string* e i valori sono di tipo *object* e rappresentano gli oggetti da memorizzare. Essa permette, quindi, di impostare e recuperare risorse, come per esempio stili e colori.

Oltre a queste proprietà, la classe `Application` definisce i metodi per gestire il ciclo di vita dell'app. Attraverso l'utilizzo di alcuni metodi possiamo definire le azioni che la nostra app deve compiere in una determinata circostanza o stato.

Gli stati di un'app possono essere riassunti nel seguente modo:

- *Not Running*: l'app non è stata ancora avviata.
- *Running*: L'app è in esecuzione e visibile sullo schermo dello smartphone
- *Pause/Inactive*: l'app è interrotta da qualche evento esterno, come ad esempio una telefonata.
- *Backgrounded*: l'app è in background (non visibile sullo schermo), ma sottintende comunque delle operazioni in corso.
- *Idle/Suspended*: l'app non sta eseguendo nessuna operazione e viene sospesa dal sistema operativo; quest'ultimo manterrà in vita il processo dell'app.

Essendo iOS, Android e Windows Phone sistemi operativi molto diversi, questi gestiscono in maniera differente gli stati delle app; saranno, quindi, diversi i metodi che gestiscono il comportamento dell'app. Gli stati nei diversi sistemi operativi sono schematizzati nella Figura 4.4.

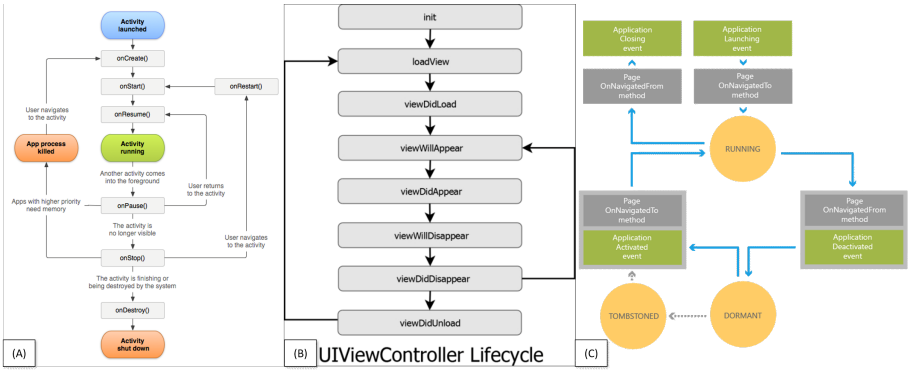

Figura 4.4. Ciclo di vita di: (a) un'*Activity* di Android, (b) un *View Controller* di iOS, e (c) una *Page* di Windows Phone

Xamarin.Forms risolve la discordanza tra queste tre modalità unificandole e definendo nuovi metodi, facilitando notevolmente lo sviluppo di app cross-platform.

Gli stati di un'applicazione Xamarin.Forms sono regolati dai seguenti metodi:

- `OnStart`: È chiamato quando l'app viene inizializzata e, in generale, è visualizzata sullo schermo.
- `OnSleep`: È richiamato quando l'app viene sospesa ed è in background.
- `OnResume`: È chiamato quando l'app è nuovamente in foreground, dopo essere stata messa in background.

In Xamarin.Forms, come si può notare, non c'è un metodo di terminazione; questo stato, infatti, viene gestito a partire dal metodo `OnSleep`. Sarà questo, l'ultimo metodo chiamato prima della fine dell'esecuzione dell'app; dopo la sua chiamata non abbiamo alcuna notifica che ci avverta di un'eventuale terminazione dell'app.

In Xamarin.Forms la creazione di una pagina è gestita dal suo costruttore [1].

4.1.4 La scrittura del log in Xamarin

Lo strumento log di debug fornisce un modo per visualizzare l'output del log durante il debug di un'app tramite Visual Studio. É uno strumento piuttosto comune a cui ricorrono gli sviluppatori per il debug delle proprie applicazioni, esso viene attraverso le chiamate del metodo `Console.WriteLine`. Nel Listato 4.1 sono riportati alcuni esempi di chiamata a questo metodo.

```
string tag = "myapp";

Log.Info (tag, "this is an info message");
Log.Warn (tag, "this is a warning message");
Log.Error (tag, "this is an error message");
```

Listing 4.1. É possibile aggiungere un tag in modo da riconoscere più facilmente la provenienza del log.

Tuttavia nelle piattaforme mobile, come Android, non è presente una console, ma è possibile scrivere `log` leggibili. Visual Studio, infatti, fornisce uno strumento, capace di leggere l'output del `log` di debug, chiamato "logcat". Questo strumento è supportato su telefoni, tablet e dispositivi indossabili Android fisici, ma anche su dispositivi virtuali, in esecuzione nell'emulatore Android.

4.1.5 Il testing su dispositivo

Come già anticipato è possibile eseguire il debug della nostra app Xamarin su dispositivi fisici come smartphone, tablet e dispositivi indossabili.

In particolare, è possibile utilizzare uno smartphone Android attivando la modalità sviluppatore su di esso, dando così il permesso per l'utilizzo del debug usb.

Una volta che il dispositivo è connesso al computer, per poter effettuare il debugging di un'applicazione Xamarin, basterà selezionare il dispositivo fisico, nella finestra in alto a destra nell'interfaccia dell'IDE, come riportato nella Figura 4.5.

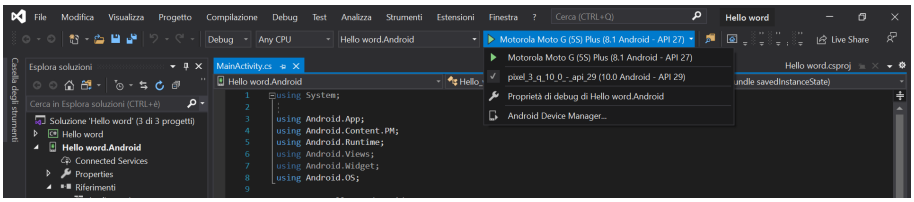

Figura 4.5. Selezionando il dispositivo fisico connesso via usb, sarà avviata la nostra app su di esso

L'emulatore Android può essere eseguito in diverse configurazioni, per simulare dispositivi Android differenti e testare la nostra app in maniera più completa.

É possibile selezionare una serie di alternative. Alla fine di questo processo viene creato un dispositivo virtuale con le caratteristiche del dispositivo fisico su cui vogliamo testare la nostra app.

É necessario selezionare il dispositivo creato nel menù a cascata in alto a destra, cliccando sul pulsante debug. Sarà, quindi, possibile interagire con il dispositivo virtuale dalla finestra dedicata, come nella Figura 4.6.

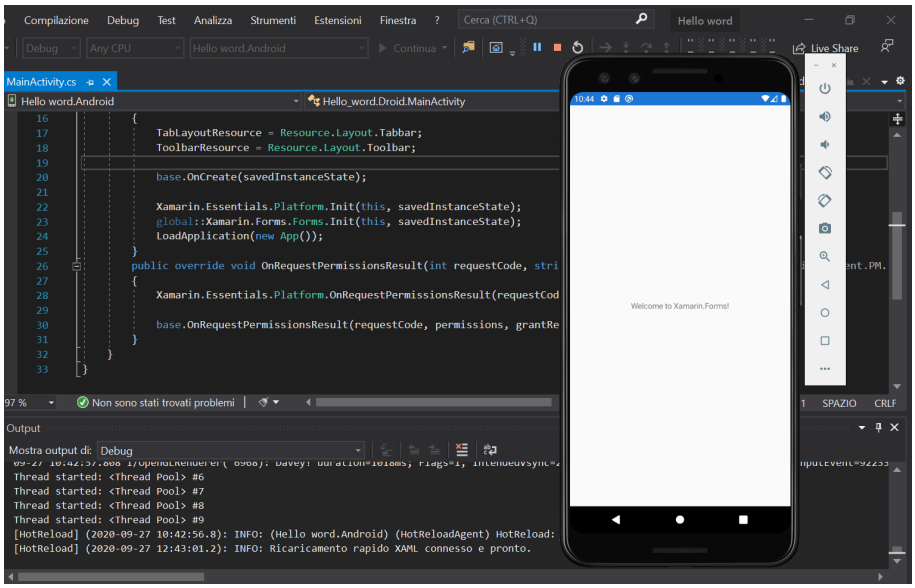

Figura 4.6. Selezionando l'emulatore precedentemente impostato, avvieremo la nostra app su di esso

4.1.6 Interfaccia dell'ambiente di sviluppo

Visual Studio è un IDE moderno, che fornisce una serie di strumenti utili per facilitare il lavoro dello sviluppatore.

Nella Figura 4.7 vediamo una finestra di lavoro di Visual Studio con un progetto aperto e varie finestre degli strumenti di base:

- *Esplora soluzioni* (in alto a destra): consente di visualizzare, esplorare e gestire i file e le cartelle che compongono la nostra soluzione. Esso consente, quindi, di organizzare il codice raggruppando i file in soluzioni e progetti.
- L'*Editor* (al centro): è senza dubbio lo strumento più utilizzato e ha il compito di visualizza il contenuto dei file. Nella finestra è possibile modificare il codice o progettare un'interfaccia utente, interagendo con l'apposito strumento.
- *Team Explorer* (in basso a destra): consente di tenere traccia e di gestire il codice condiviso con altri utenti, usando sistemi di controllo di versione (CSV), come Git e Team Foundation Version Control (TFVC).

Le funzionalità più note di Visual Studio, che permettono una maggiore produttività nello sviluppo del software, includono:

- *Controllo dell'ortografia durante la digitazione e azioni rapide.* Il controllo dell'ortografia durante la digitazione visualizza sottolineature ondulate di colore

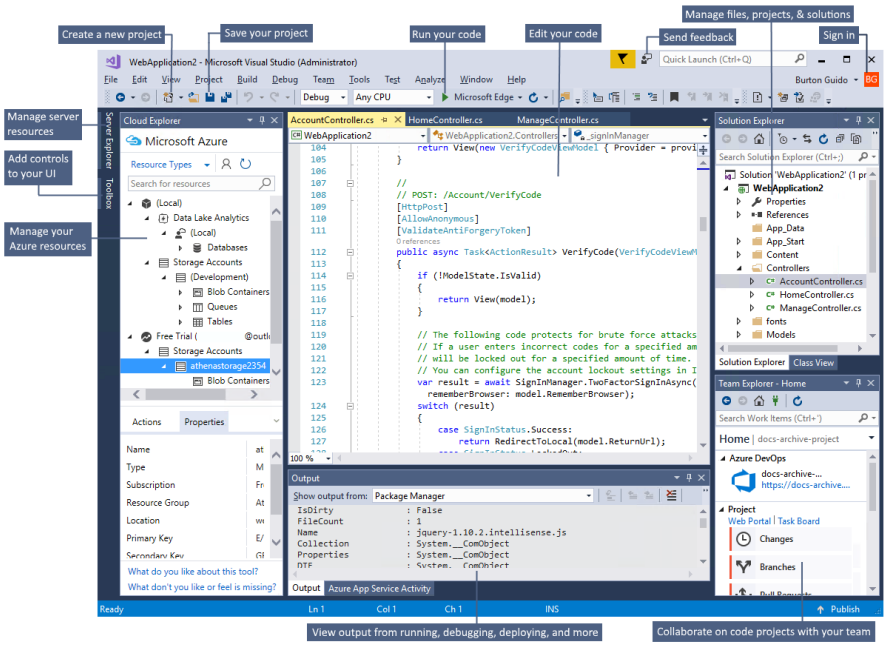

Figura 4.7. Breve panoramica dell'interfaccia di Visual Studio

rosso, che segnalano errori veri o potenziali nel codice in tempo reale. Queste segnalazioni consentono di risolvere i problemi immediatamente, senza attendere che l'errore venga rilevato durante la compilazione o l'esecuzione del codice.
Passando il mouse su una linea ondulata vengono visualizzate informazioni aggiuntive riguardo l'errore. Sul margine sinistro può essere visualizzata anche una lampadina con le "azioni rapide" utili per risolvere il problema segnalato. É riportato un esempio nella Figura 4.8.

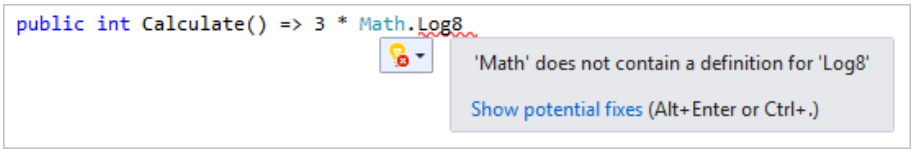

Figura 4.8. Nell'immagine viene riportato un esempio di errore di battitura in fase di stesura del codice

- *Pulizia del codice.* Con il semplice click di un pulsante è possibile formattare il codice e applicare eventuali correzioni suggerite dalle impostazioni di stile, dalle convenzioni `.editorconfig` e dagli analizzatori Roslyn.
  Lo strumento di pulizia del codice consente di risolvere i problemi nel codice, prima di passare alla,sua revisione del codice. É riportato un esempio nella Figura 4.9.

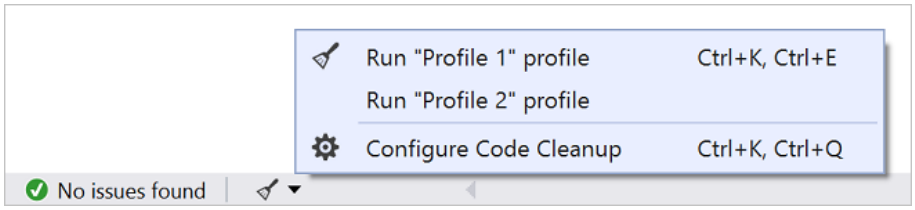

Figura 4.9. Nell'immagine un esempio di utilizzo dello strumento di pulizia del codice, selezionabile nella parte inferiore della nostra finestra

- *Refactoring del codice.* Il refactoring prevede operazioni come la ridenominazione delle variabili mantenendo la coerenza in tutto il progetto, l'estrazione di una o più righe di codice in un nuovo metodo, la modifica dell'ordine dei parametri dei metodi, e altro ancora. Nella Figura 4.10 viene riportato un esempio di utilizzo di questo strumento.

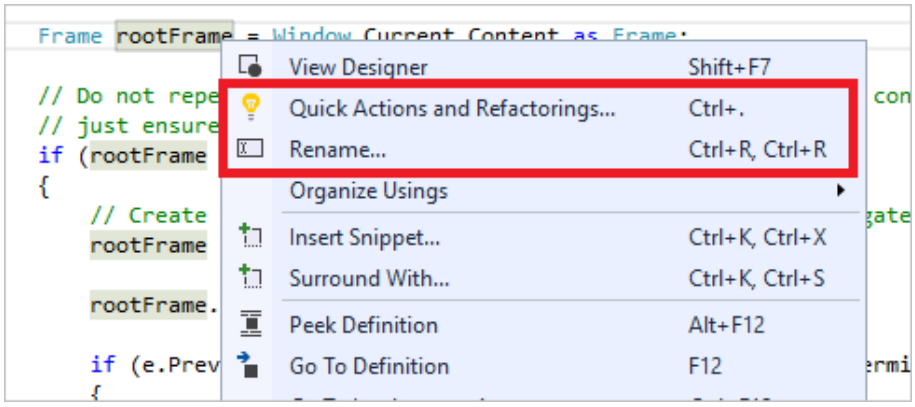

Figura 4.10. Nell'immagine un esempio di utilizzo dello strumento di refactoring del codice

- *IntelliSense.* IntelliSense è, a sua volta, composto da una serie di strumenti utili per velocizzare la scrittura del codice della nostra app, come il completamento automatico di piccole parti del codice. È come se si avesse a disposizione la documentazione di base all'interno dell'editor, senza dover cercare le informazioni sul tipo altrove. La Figura 4.11 mostra come IntelliSense visualizza un elenco di membri per un tipo.
- *XAML Previewer.* Con XAML Previewer sarà possibile visualizzare un'interfaccia utente, senza dover compilare ed eseguire la nostra app, velocizzando, in questo modo, notevolmente, i tempi di sviluppo e di testing. Questo strumento è facilmente raggiungibile sul lato destro del nostro editor, come riportato in Figura 4.12.
- *Casella di ricerca.* La casella di ricerca fornisce la possibilità di ricercare rapidamente quello che ci serve in Visual Studio. Essa permette di trovare ciò che ci interessa, per esempio all'interno del nostro progetto, all'interno delle imposta-

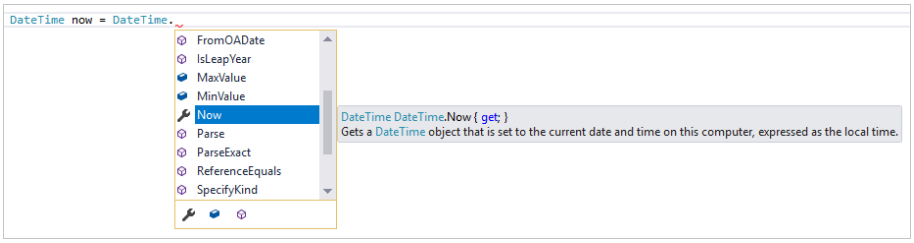

Figura 4.11. Nell'immagine un esempio di utilizzo di IntelliSense

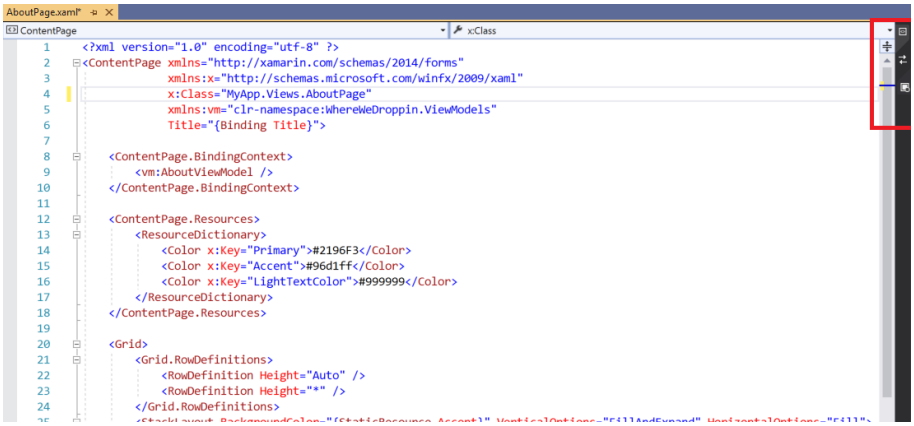

Figura 4.12. Nell'immagine un esempio di utilizzo di XAML Previewer

zioni dell'IDE, oppure, tra i pacchetti disponibili per essere aggiunti al nostro progetto. La Figura 4.13 riporta un esempio di ricerca in Visual Studio.

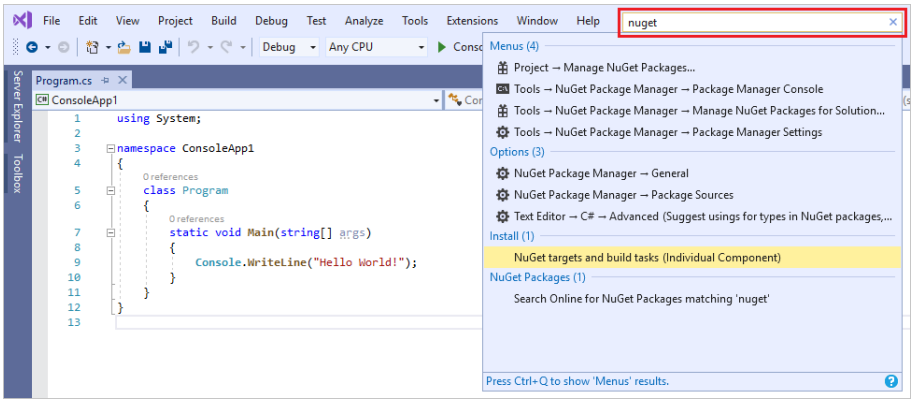

Figura 4.13. Nell'immagine un esempio di ricerca effettuata con strumento lo fornito da Visual Studio

- *Live Share.* Con Live Share è possibile apportare modifiche ed eseguire il debug

con altri utenti in tempo reale. Si può condividere il progetto in modo immediato e sicuro e, in base alle esigenze, si possono condividere sessioni di debugging o istanze del terminale.

- *Gerarchia delle chiamate.* La finestra delle gerarchie di chiamata mostra i punti del codice in cui il metodo selezionato viene chiamato. Si tratta di informazioni che possono risultare utili quando si prevede di cambiare o rimuovere il metodo o quando si tenta di risolvere un bug. Nella Figura 4.14 è presente un esempio di utilizzo di questo strumento.

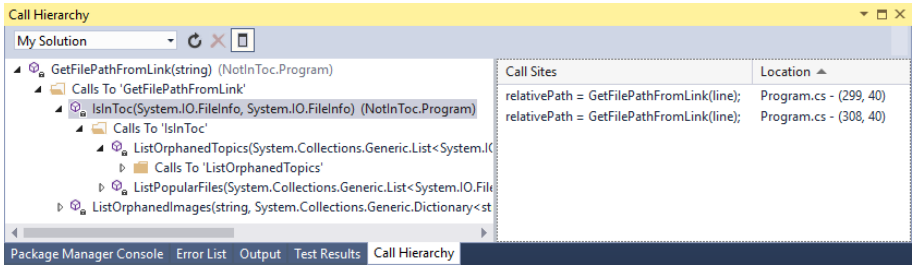

Figura 4.14. Nella barra in basso nel nostro IDE è possibile aprire una finestra in cui visualizzare l'intera gerarchia di una chiamata a un metodo

- *CodeLens.* Esso è uno strumento che consente di trovare i riferimenti, le modifiche al codice, i bug collegati, gli elementi di lavoro, le revisioni del codice e gli `unit test`, senza uscire dall'editor, come riportato nella Figura 4.15.

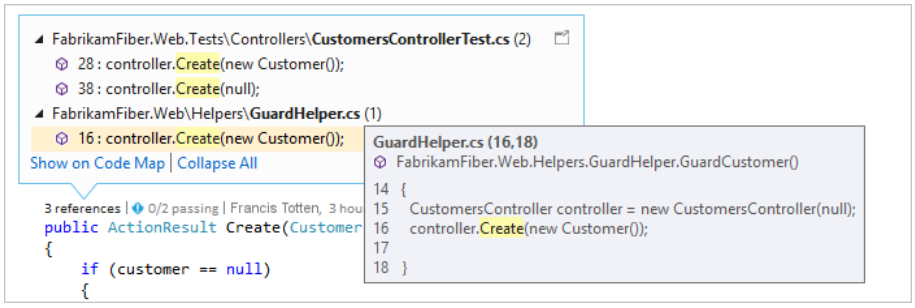

Figura 4.15. CodeLens ci fornisce una breve panoramica della porzione di codice di nostro interesse

- *Vai a definizione.* Questa funzionalità consente di passare direttamente alla porzione di codice in cui vengono definiti una funzione o un tipo utilizzato. Cliccando su un metodo o un tipo con il pulsante destro del mouse, apparirà una finestra simile a quella riportata nella Figura 4.16 [2] .

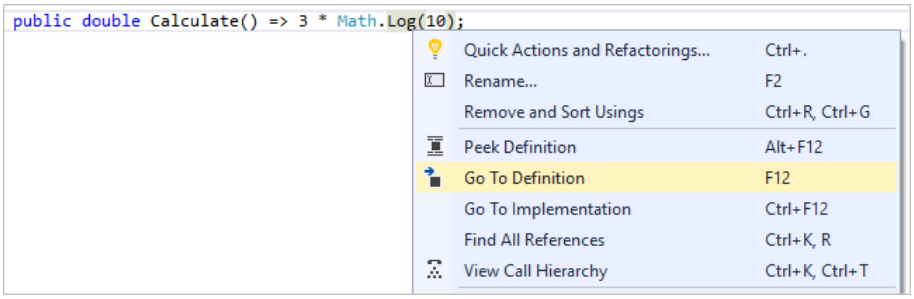

Figura 4.16. Alcune delle opzioni che l'IDE ci fornisce per interagire con i metodi e i tipi del nostro progetto

4.1.7 Documentazione

Microsoft fornisce un'ampia documentazione che affianca lo sviluppatore dall'installazione fino all'implementazione di specifici metodi di Xamarin. Sono presenti, inoltre, molti esempi ben commentati, che spaziano dagli elementi più semplici, come, ad esempio, l'interfaccia utente, fino ad arrivare ai meccanismi più complessi del framework. Queste risorse sono raggiungibili dal sito *docs.microsoft.com/it-it/xamarin/xamarin-forms/*, la cui home page viene riportata nella Figura 4.17.

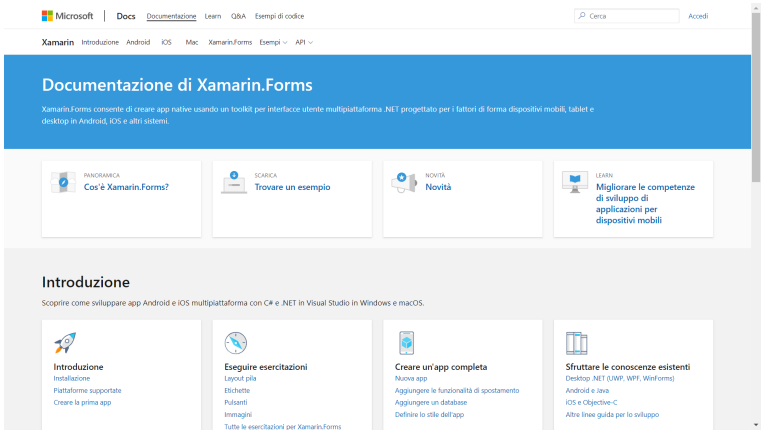

Figura 4.17. Home page del sito *docs.microsoft.com/it-it/xamarin/xamarin-forms/*

4.2 Interfaccia in Xamarin

Xamarin.Forms può essere, di fatto, considerata come un livello di astrazione per lo sviluppo di UI cross-platform.

Questa tecnologia permette di utilizzare una base di codice back-end condivisa tra le piattaforme e, allo stesso tempo, di avere un'interfaccia utente del tutto nativa.

Ogni interfaccia UI sarà definita da un file in C# e un file in XAML.

XAML (abbreviazione di eXtensible Application Markup Language) è un linguaggio dichiarativo di markup basato su XML sviluppato da Microsoft. Esso, fa parte dell'ecosistema .NET e permette di descrivere le UI delle nostre app.

Gli elementi della sintassi XAML rappresentano direttamente le istanze degli oggetti, tuttavia è possibile interagire con tali risorse attraverso il code-behind e, quindi, definire il comportamento nel codice C# associato.

Xamarin fornisce alcune alternative per la definizione di interfacce utente; queste verranno descritte nelle sottosezioni seguenti.

4.2.1 Le interfacce utente di Xamarin nei diversi sistemi operativi

Utilizzando Xamarin, si possono sviluppare interfacce utente (UI) cross-platform, fornendo anche la possibilità di sviluppare elementi specifici per l'app di una determinata piattaforma.

Xamarin fornisce due approcci alternativi per costruire UI:

- *Xamarin.Forms*: è una libreria cross-platform (Android, iOS) per la UI.
- *Platform-specific* (native UI): esso consente lo sviluppo attraverso l'utilizzo di Xamarin.Android e Xamarin.iOS.

É possibile adattare delle specifiche porzioni di codice cross-platform ad un sistema operativo specifico. Ciò può avvenire attraverso la scrittura sia del codice XAML che di quello C#, come riportato, rispettivamente, nei Listati 4.2 e 4.3

```
1  <ContentPage>
2      <ContentPage.Padding>
3          <OnPlatform x:TypeArguments="Thickness">
4              <On Platform="iOS, Android" Value="0,40,0,0" />
5          </OnPlatform>
6      </ContentPage.Padding>
7      [...]
8  </ContentPage>
```

Listing 4.2. Esempio di codice XAML in cui un elemento UI viene definito in maniera differente per diversi sistemi operativi all'interno di un `tag`

```
1  switch (Device.RuntimePlatform)
2          {
3              case Device.iOS:
4              case Device.Android:
5                  padding = new Thickness(0, 40, 0, 0);
6                  break;
7              default:
8                  padding = new Thickness();
9                  break;
10         }
```

Listing 4.3. Esempio di codice C# in cui un elemento UI viene definito in maniera differente per diversi sistemi operativi all'interno di uno `switch/case`

4.2.2 Definizione degli elementi dell'UI nel file XAML e nel file C#

In questa sezione verrà mostrato un esempio di pulsante (`button`); in particolare, verranno confrontate due alternative, per la realizzazione della UI fornite da Xamarin.

Esempio di UI creata da codice XAML

Nel Listato 4.4 viene riportato un esempio di file `XAML` in cui viene definito uno `StackLayout` all'interno del quale sono definiti una `Label` e un `Button`.

```
1   <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
2                xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
3                x:Class="ButtonDemos.BasicButtonClickPage"
4                Title="Basic Button Click">
5       <StackLayout>
6
7           <Label x:Name="label"
8                  Text="Click the Button below"
9                  FontSize="Large"
10                 VerticalOptions="CenterAndExpand"
11                 HorizontalOptions="Center" />
12
13          <Button Text="Click to Rotate Text!"
14                  VerticalOptions="CenterAndExpand"
15                  HorizontalOptions="Center"
16                  Clicked="OnButtonClicked" />
17
18      </StackLayout>
19  </ContentPage>
```

Listing 4.4. Esempio di UI definita in XAML

Come anticipato, in Xamarin le UI sono definite da un file XAML e uno C#; in quest'ultimo è possibile definire il codice che interagirà con gli elementi definiti dell'interfaccia.

Nel Listato 4.5 viene riportato il code-behind del nostro esempio, in esso è implementato il metodo di gestione del click del pulsante `OnButtonClicked()`.

```
1   public partial class BasicButtonClickPage : ContentPage
2   {
3       public BasicButtonClickPage ()
4       {
5           InitializeComponent ();
6       }
7
8       async void OnButtonClicked(object sender, EventArgs args)
9       {
10          await label.RelRotateTo(360, 1000);
11      }
12  }
```

Listing 4.5. Questo gestore si trova nel file code-behind

Esempio di UI creata da codice C#

Nel Listato 4.6 viene illustrato come creare una pagina equivalente dal punto di vista funzionale al listato precedente; in questo caso, però, il codice è scritto in C#, utilizzando una sintassi dichiarativa.

Come si può vedere nell'esempio, lo `StackLayout` verrà popolato sfruttando la sua proprietà `Children`.

```
1   public class CodeButtonClickPage : ContentPage
2   {
3       public CodeButtonClickPage ()
4       {
5           Title = "Code Button Click";
6
7           Label label = new Label
8           {
9               Text = "Click the Button below",
10              FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label)),
11              VerticalOptions = LayoutOptions.CenterAndExpand,
12              HorizontalOptions = LayoutOptions.Center
13          };
14
15          Button button = new Button
16          {
17              Text = "Click to Rotate Text!",
18              VerticalOptions = LayoutOptions.CenterAndExpand,
19              HorizontalOptions = LayoutOptions.Center
20          };
21          button.Clicked += async (sender, args) => await label.RelRotateTo(360, 1000);
22
23          Content = new StackLayout
24          {
25              Children =
26              {
27                  label,
28                  button
29              }
30          };
31      }
32  }
```

Listing 4.6. Esempio di UI definita in codice C#

Ci sono alcuni vantaggi e alcuni svantaggi derivanti da ciascuna delle due opzioni; nella pratica, non è esclusa la coesistenza dei due approcci.

Scrivendo le UI in XAML, sicuramente avremo un metodo più veloce e di più semplice interpretazione; esso sarà, per esempio, estremamente utile per definire elementi statici della UI. Inoltre potrebbe essere più semplice da utilizzare qualora un programmatore conoscesse già linguaggi di markup come HTML.

Definire le UI in C# risulterà più utile qualora si volessero implementare elementi popolati in maniera iterativa, per esempio sfruttando un `foreach`.

4.2.3 Esempi interfaccia

Nelle sottosezioni seguenti verranno descritti alcuni listati derivati dal progetto oggetto descritti nella presente tesi. In particolare, verranno riportati esempi della UI dell'app e della logica sottostante, tra cui le chiamate alle API REST.

MainPage

La `MainPage` è la prima pagina visibile nella nostra app. Essendo una pagina statica molto semplice è stata implementata completamente in XAML.

Nel Listato 4.7 é stato riportato il codice C# del file `MainPage.cs`, in cui viene semplicemente istanziata la `ContentPage`.

```
1   using Xamarin.Forms;
2
3   namespace FoodXamarin
4   {
5       public partial class MainPage : ContentPage
6       {
7           public MainPage()
8           {
9               InitializeComponent();
10          }
11      }
```

```
12  }
```

Listing 4.7. Codice di `MainPage.cs`

Nel Listato 4.8 è riportato il codice XAML del file `MainPage.xaml`.

Nella parte superiore della schermata viene definita una `Grid`, con sfondo azzurro; al centro di questa sono visualizzati il logo e il nome dell'app.

Nella parte centrale della schermata, invece, sono visualizzate due `Label`.

```
1   <?xml version="1.0" encoding="utf-8" ?>
2   <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
3                xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
4                xmlns:d="http://xamarin.com/schemas/2014/forms/design"
5                xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6                mc:Ignorable="d"
7                x:Class="FoodXamarin.MainPage">
8
9       <ContentPage.Resources>
10          <ResourceDictionary>
11              <Color x:Key="Primary">#2196F3</Color>
12              <Color x:Key="Accent">#96d1ff</Color>
13              <Color x:Key="LightTextColor">#999999</Color>
14          </ResourceDictionary>
15      </ContentPage.Resources>
16
17      <Grid>
18          <Grid.RowDefinitions>
19              <RowDefinition Height="Auto" />
20              <RowDefinition Height="*" />
21          </Grid.RowDefinitions>
22          <StackLayout BackgroundColor="{StaticResource Accent}" VerticalOptions="FillAndExpand"
            HorizontalOptions="Fill">
23              <StackLayout Orientation="Horizontal" HorizontalOptions="Center" VerticalOptions="Center">
24                  <ContentView Padding="0,40,0,40" VerticalOptions="FillAndExpand">
25                      <Grid>
26                          <Grid.ColumnDefinitions>
27                              <ColumnDefinition Width="Auto"/>
28                          </Grid.ColumnDefinitions>
29                          <Image Grid.Row="0" Grid.Column="0" Source="food_logo_white.png"  HeightRequest="80"
            HorizontalOptions="Center" />
30                          <Label Grid.Row="0" Grid.Column="1" Text="Food" TextColor="White" FontSize="20"
            VerticalOptions="Center" HorizontalOptions="Center"/>
31                      </Grid>
32                  </ContentView>
33              </StackLayout>
34          </StackLayout>
35          <ScrollView Grid.Row="1">
36              <StackLayout Orientation="Vertical" Padding="16,40,16,40" Spacing="10">
37
38                  <Label HorizontalOptions="CenterAndExpand">
39                      <Label.FormattedText>
40                          <FormattedString>
41                              <FormattedString.Spans>
42                                  <Span Text="Benvenuto in " FontSize="24"/>
43                                  <Span Text="Food" FontAttributes="Bold" FontSize="26" />
44                                  <Span Text="." FontSize="24"/>
45                              </FormattedString.Spans>
46                          </FormattedString>
47                      </Label.FormattedText>
48                  </Label>
49              </StackLayout>
50          </ScrollView>
51      </Grid>
52  </ContentPage>
```

Listing 4.8. Codice di `MainPage.xaml`

MasterDetail

Per navigare tra le diverse pagine dell'app, è stata implementato un menù di tipo `MasterDetail`.

Nel file `MDPage.xaml`, riportato nel Listato 4.9, sono definite la pagina *Master*, all'interno del tag XAML `<MasterDetailPage.Master>`, e la pagina *Detail*, all'interno del tag `<MasterDetailPage.Detail>`.

All'interno del *Master* sono definiti i *button* che, all'evento `Clicked`, chiameranno le opportune funzioni definite in `MDPage.cs`.

La sezione *Detail* è lasciata vuota; verrà, infatti, riempita con la pagina da visualizzare in base all'opzione selezionata dall'utente.

```
<?xml version="1.0" encoding="utf-8" ?>
<MasterDetailPage xmlns="http://xamarin.com/schemas/2014/forms"
                  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                  xmlns:d="http://xamarin.com/schemas/2014/forms/design"
                  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
                  mc:Ignorable="d"
                  x:Class="FoodXamarin.Views.MDPage">
    <MasterDetailPage.Master>
        <ContentPage Padding="10" BackgroundColor="AliceBlue" Title="Master">
            <ContentPage.Content>
                <StackLayout Margin="5, 30, 5, 5">
                    <Label Text="Food">
                        </Label>
                        <Button Text="Menu" Clicked="MenuButton_Clicked">
                        </Button>
                    <Button Text="Prenota un tavolo" Clicked="PrenotaTavolo_clicked">
                    </Button>
                    <Button Text="About" Clicked="AboutButton_Clicked">
                    </Button>
                </StackLayout>
            </ContentPage.Content>
        </ContentPage>
    </MasterDetailPage.Master>


    <MasterDetailPage.Detail>
        <ContentPage Padding="10">
            <ContentPage.Content>
                <StackLayout Margin="5, 30, 5, 5">
                    <Label Text="Detail Page">

                    </Label>
                </StackLayout>
            </ContentPage.Content>
        </ContentPage>
    </MasterDetailPage.Detail>
</MasterDetailPage>
```

Listing 4.9. Codice di `MDPage.xaml`

Nel file `MDPage.cs` sono implementati i metodi che, invocati dall'evento click dei *button*, impostano il contenuto della finestra corretta, istanziando una `new NavigationPage()`.

```
using System;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace FoodXamarin.Views
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class MDPage : MasterDetailPage
    {
        public MDPage()
        {
            InitializeComponent();
            Detail = new NavigationPage(new MainPage());
        }
        private void AboutButton_Clicked(object sender, EventArgs e)
        {
            Detail = new NavigationPage(new About());
            IsPresented = false;
        }

        private void PrenotaTavolo_clicked(object sender, EventArgs e)
        {
            Detail = new NavigationPage(new PrenotazioneTavoloView());
            IsPresented = false;
        }

        private void MenuButton_Clicked(object sender, EventArgs e)
        {
            Detail = new NavigationPage(new MenuView());
            IsPresented = false;
        }
    }
}
```

Listing 4.10. Codice di `MDPage.cs`

MenuView

Nella schermata `MenuView` sarà visibile il menù del ristorante. Per fare ciò nella classe `MenuView.cs`, riportata nel Listato 4.11, viene implementata una *ViewCell*, chiamata *ProdottoCell*, all'interno della quale sono definite tutte le caratteristiche delle *label*, che verranno poi popolate con i dati di nostro interesse.

Il metodo `getProdotti()`, implementato in `MenuController.cs`, effettuerà una chiamata al server e, successivamente, attraverso il metodo `fillListView`, popolerà le *label*, poi visualizzate nella nostra interfaccia.

Le `label` sono inserite in una *Grid*, attraverso il metodo `grid.Children.Add()`; questo riceverà in input la *label* e la posizione specifica di questa da assegnare all'interno della *Grid*. La *Grid* darà, quindi, la struttura della singola riga della nostra *ListView*.

Al click dell'elemento della *ListView* verrà chiamato il metodo `void` asincrono `OnActionProdClicked()`, all'interno del quale verrà chiamato il metodo `DisplayActionSheet` che farà apparire sullo schermo una finestra di popup, per permettere all'utente di controllare la scelta del prodotto e di inserire il numero di porzioni desiderate.

Al click sul pulsante "Conferma", verrà, poi, visualizzata un'altra finestra di dialogo, attraverso l'invocazione del metodo `DisplayActionSheet`, nella quale sarà possibile concludere la selezione del prodotto.

```
1   using FoodXamarin.Model;
2   using FoodXamarin.Controller;
3   using System.Collections.Generic;
4   using System.Diagnostics;
5   using Xamarin.Forms;
6   using Xamarin.Forms.Xaml;
7
8   namespace FoodXamarin.Views
9   {
10      [XamlCompilation(XamlCompilationOptions.Compile)]
11      public partial class MenuView : CarouselPage
12      {
13
14          public MenuView()
15          {
16              InitializeComponent();
17
18              MenuController request = new MenuController(this);
19              request.getProdotti();
20          }
21
22          private void AddProdotto(int prodottoId, int clienteId)
23           {
24              MenuController request = new MenuController(this);
25              request.addOrdine(prodottoId, clienteId);
26          }
27
28
29          public void fillListView(List<Prodotto> response)
30          {
31              lista_prodotti.ItemsSource = response;
32              lista_prodotti.ItemTemplate = new DataTemplate(typeof(ProdottoCell));
33          }
34
35          class ProdottoCell : ViewCell
36          {
37              public ProdottoCell()
38              {
39                  Label idLabel = new Label
40                  {
41                      FontSize = 15,
42                      HorizontalOptions = LayoutOptions.CenterAndExpand
43                  };
44                  Label nomeLabel = new Label
45                  {
```

```
46                      FontSize = 20,
47                      TextColor = Color.Black,
48                      HorizontalOptions = LayoutOptions.StartAndExpand,
49                      VerticalOptions = LayoutOptions.FillAndExpand
50                  };
51                  Label descrizioneLabel = new Label
52                  {
53                      FontSize = 15,
54                      HorizontalOptions = LayoutOptions.Start,
55                      VerticalOptions = LayoutOptions.End
56                  };
57                  Label prezzoLabel = new Label
58                  {
59                      FontSize = 20,
60                      TextColor = Color.Black,
61                      HorizontalOptions = LayoutOptions.End,
62                      VerticalOptions = LayoutOptions.Start,
63                      Padding = new Thickness(5, 5, 5, 5),
64                  };
65                  Label euro = new Label
66                  {
67                      Text = "€"
68                  };
69
70                  idLabel.SetBinding(Label.TextProperty, "id");
71                  nomeLabel.SetBinding(Label.TextProperty, "nome");
72                  descrizioneLabel.SetBinding(Label.TextProperty, "descrizione");
73                  prezzoLabel.SetBinding(Label.TextProperty, "prezzo");
74                  categoriaLabel.SetBinding(Label.TextProperty, "categoria_id");
75
76                  Grid grid = new Grid
77                  {
78                      ColumnSpacing = 0,
79                      RowDefinitions =
80                      {
81                          new RowDefinition(),
82                      },
83                      ColumnDefinitions =
84                      {
85                          new ColumnDefinition(){
86                              Width = new GridLength(180)
87                          },
88                          new ColumnDefinition(),
89                          new ColumnDefinition
90                          {
91                              Width = new GridLength(50)
92                          }
93                      }
94                  };
95
96                  grid.Children.Add(nomeLabel, 0, 0);
97                  grid.Children.Add(prezzoLabel, 1, 0);
98                  grid.Children.Add(descrizioneLabel, 0, 1);
99                  grid.Children.Add(euro, 2, 0);
100
101                 View = grid;
102             }
103         }
104
105         async void OnActionProdClicked(object sender, ItemTappedEventArgs e)
106         {
107             var layout = (BindableObject)sender;
108             Prodotto selprod = (Prodotto)e.Item;
109
110             string result = await DisplayPromptAsync(
111                 "Seleziona numero di porzioni \r\n di " + selprod.nome,
112                 "\r\n Descrizione: "+ selprod.descrizione +
113                 "\r\n prezzo: " +selprod.prezzo+ " €",
114                 accept: "Conferma", cancel: "Annulla", initialValue: "1", maxLength: 2, keyboard:
        Keyboard.Numeric);
115             Debug.WriteLine("result: "+ result);
116             if (result != null)
117             {
118                 string action = await DisplayActionSheet(
119                     "Conferma", "Conferma","Annulla", "Vuoi ordinare " + result
120                     + " porzioni di " + selprod.nome + " ?");
121                 int idProdotto = selprod.id;
122                 Debug.WriteLine("Action: " + action);
123                 Debug.WriteLine("id prodotto: " + idProdotto);
124             };
125         }
126     }
127 }
```

Listing 4.11. Codice di `MenuView.cs`

Nel file `MenuView.xaml` riportato nel Listato 4.12, sono definiti sia gli elementi statici della UI, che quelli dinamici; in particolare, verrà definita una *ListView* con nome *lista_prodotti*, che permetterà di visualizzare la lista dei prodotti nel

menù. Inoltre, all'interno del tag *ListView*, sarà definito il metodo associato al click dell'utente.

```
<?xml version="1.0" encoding="utf-8" ?>
<CarouselPage xmlns="http://xamarin.com/schemas/2014/forms"
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
              xmlns:d="http://xamarin.com/schemas/2014/forms/design"
              xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
              mc:Ignorable="d"
              x:Class="FoodXamarin.Views.MenuView"
              Title="Menu">
    <ContentPage>

        <StackLayout
        HorizontalOptions="FillAndExpand"
        VerticalOptions="FillAndExpand"
        Padding="5,5,5,5">

            <Label HorizontalOptions="CenterAndExpand">
                <Label.FormattedText>
                    <FormattedString>
                        <FormattedString.Spans>
                            <Span Text="Primi piatti" FontSize="22"/>
                        </FormattedString.Spans>
                    </FormattedString>
                </Label.FormattedText>
            </Label>
            <ListView x:Name="lista_prodotti"
                  ItemTapped="OnActionProdClicked"
                  HorizontalOptions="FillAndExpand"
                  VerticalOptions="FillAndExpand"
                  RowHeight="80"/>
        </StackLayout>
    </ContentPage>
</CarouselPage>
```

Listing 4.12. Codice di `MenuView.xaml`

4.3 Librerie utilizzate

Nella soluzione è stato necessario implementare un meccanismo di chiamate alle API REST. In particolare, per fare ciò, sono state utilizzate le librerie `Http.Client` e `Newtonsoft`.

La prima serve per per l'invio di richieste `HTTP` e la ricezione delle corrispettive risposte.

La seconda è utile per manipolare oggetti `JSON`, cioè per serializzare e deserializzare gli oggetti del nostro progetto, in maniera semplice e veloce, attraverso `Json.NET`.

Entrambe le librerie fanno parte dell'ecosistema .NET, ed entrambe sono necessarie per interagire con le API REST sviluppate per il nostro progetto.

Nella prossima sottosezione verrà riportato un esempio di utilizzo di queste due librerie.

4.3.1 Esempio di chiamata Json in MenuController.cs

Nel Listato 4.13 è riportata la classe `MenuController.cs`, utile per comprendere le modalità di interazione con le `API` e con gli oggetti `JSON`.

Come primo passo bisogna permettere alla nostra classe di utilizzare le librerie di nostro interesse; per fare ciò è necessario inserire le righe `using Newtonsoft.Json` e `using System.Net.Http`. In questo modo saranno aggiunti i due *namespace* alla

nostra classe, permettendo, quindi, la chiamata al suo intero, di tutti i metodi presenti nel *namespace* dichiarato.

Successivamente, nel costruttore della classe, vengono definiti l'*URL* e il tipo di *client* utilizzati.

Attraverso la chiamata `this.URL = "http://188.166.19.9/api/prodotti"` si inserisce l'indirizzo IP che identificano le `API` del nostro progetto e la risorsa a cui vogliamo mandare la richiesta, in questo caso "prodotti" .

Attraverso la riga `this._client = new HttpClient()` definiamo il protocollo di comunicazione utilizzato.

Analizzando il caso del metodo `getProdotti()`, completiamo il nostro esempio di chiamata `JSON`; in particolare, faremo una chiamata `GET` alle nostre `API`.

Innanzitutto, questo metodo verrà chiamato direttamente dalla vista del nostro menù, come è possibile vedere nel Listato 4.11; sarà, poi, questo metodo a restituire i dati ricevuti direttamente alla vista.

Con la riga `Debug.WriteLine("getProdotti")` notifichiamo l'effettiva chiamata del metodo, attraverso la scrittura del suo nome nel `log`. Successivamente, attraverso la riga `var response = await _client.GetAsync(URL)` verrà istanziato l'oggetto *response*, chiamando il metodo asincrono `get.Async()`.

Attraverso le istruzioni condizionali `if-else` verrà, poi, individuata la porzione di codice da eseguire.

Se il server restituisce un valore diverso dal valore nullo, i dati ricevuti verranno letti, attraverso l'istruzione `var result = response.Content.ReadAsStringAsync().Result`, de-serializzati, attraverso la libreria `Newtonsoft.Json`, e inseriti in un oggetto di tipo `List<Prodotto>`, poi passato alla vista del menù.

Se il server non restituisce alcun valore, allora verrà inviata un'opportuna notifica al programmatore, attraverso il `log` con la chiamata `Debug.WriteLine("Nothing retrived from server")`.

```
1   using FoodXamarin.Model;
2   using FoodXamarin.Views;
3   using Newtonsoft.Json;
4   using System.Collections.Generic;
5   using System.Diagnostics;
6   using System.Net.Http;
7
8   namespace FoodXamarin.Controller
9   {
10      class MenuController
11      {
12          private string URL;
13          private HttpClient _client;
14          private MenuView menuView;
15
16          public MenuController(MenuView menuView)
17          {
18              this.menuView = menuView;
19              this.URL = "http://188.166.19.9/api/prodotti";
20              this._client = new HttpClient();
21          }
22
23          public async void getProdottoById(int idProdotto)
24          {
25              Debug.WriteLine("getProdotti");
26              var response = await _client.GetAsync(URL+idProdotto);
27
28              if (response.IsSuccessStatusCode)
29              {
30                  var result = response.Content.ReadAsStringAsync().Result;
31                  List<Prodotto> prodotti = JsonConvert.DeserializeObject<List<Prodotto>>(result);
32                  Debug.Write("result");
33                  Debug.Write(result);
34                  ;
35              }
36              else
37                  Debug.WriteLine("Nothing retrived from server");
38          }
```

```
39
40
41          public async void getProdotti()
42          {
43              Debug.WriteLine("getProdotti");
44              var response = await _client.GetAsync(URL);
45
46              if (response.IsSuccessStatusCode)
47              {
48                  var result = response.Content.ReadAsStringAsync().Result;
49                  List<Prodotto> prodotti = JsonConvert.DeserializeObject<List<Prodotto>>(result);
50                  Debug.Write("result");
51                  Debug.Write(result);
52
53                  menuView.fillListView(prodotti);
54                  ;
55              }
56              else
57                  Debug.WriteLine("Nothing retrived from server");
58          }
59      }
60  }
```

Listing 4.13. Codice di `MenuController.cs`

5

Manuale utente

In questo capitolo verrà descritto il manuale utente dell'app sviluppata. In particolare, verranno illustrate le schermate principali con i relativi screenshot e le modalità di utilizzo.

5.1 Schermate e spiegazione dell'utilizzo

Per il corretto funzionamento dell'applicazione sarà necessaria un collegamento internet, per mezzo di Wi-Fi oppure attraverso la connessione dati. In questo modo la nostra app potrà comunicare correttamente con i dati memorizzati sul server.

5.1.1 Pagina di login

All'avvio dell'app sarà visualizzata la schermata di login riportata nella Figura 5.1. In questa finestra l'utente potrà eseguire l'accesso, con le credenziali create durante la prima registrazione; in caso contrario, potrà registrarsi cliccando sul testo "Non sei ancora registrato? clicca qui".

Nel caso un cui l'utente avesse dimenticato i dati di accesso, potrà cliccare su "Hai dimenticato le tue credenziali? clicca qui" per risolvere il problema ed effettuare il login.

5.1.2 Prima pagina

Dopo aver effettuato il login, apparirà una prima pagina, riportata nelle Figura 5.2; in questa pagina statica sarà visualizzato il logo del ristorante con un messaggio di benvenuto.

5.1.3 Pagina Master-Detail

Come si può vedere nella Figura 5.2, cliccando in alto a sinistra, l'utente potrà aprire il menù di navigazione della nostra app. Tale menù sarà accessibile da tutte le schermate della nostra app; il suo aspetto sarà come quello riportato nella Figura 5.3.

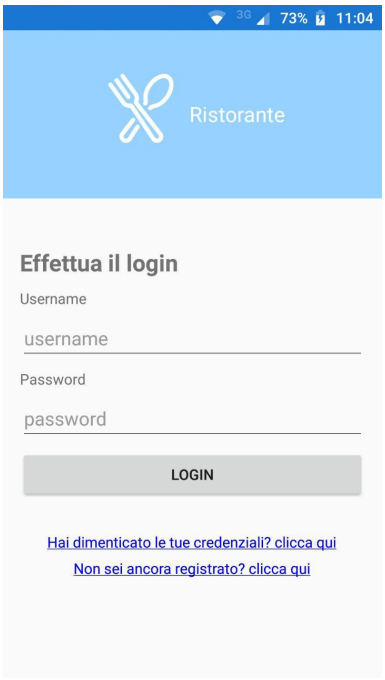

Figura 5.1. La figura riporta la finestra di accesso in cui l'utente inserirà le proprie credenziali

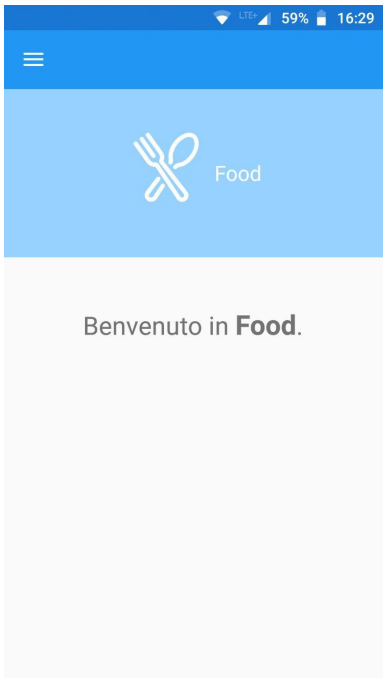

Figura 5.2. La figura riporta la prima pagina dell'app

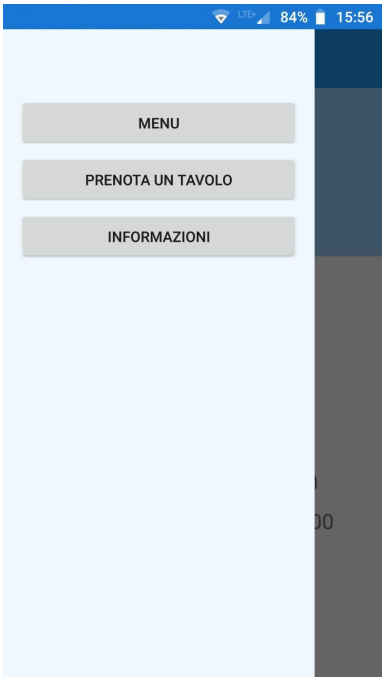

Figura 5.3. La figura riporta il menù utilizzato per navigare all'interno dell'app

5.1.4 Fare un'ordinazione

Cliccando sulla voce "Menù" si potrà accedere ai piatti proposti dal ristorante. Nella schermata sarà visibile un elenco di prodotti selezionabili, con il nome, gli ingredienti e il prezzo, come riportato nella Figura 5.4.

Dopo aver selezionato la pietanza, apparirà una finestra di dialogo, in cui l'utente potrà specificare il numero di porzioni che desidera ordinare e potrà confermare o annullare l'operazione avviata. Nella Figura 5.5 è riportata la schermata appena descritta.

Dopo aver cliccato su "Conferma" apparirà un'ulteriore finestra, in cui l'utente visualizzerà un breve riepilogo dell'ordine effettuato, come nella Figura 5.6 e potrà, controllare e confermare o annullare l'ordine.

5.1.5 Prenotare tavolo

Cliccando sulla voce "Prenota un tavolo" apparirà la finestra riportata nella Figura 5.7, in cui l'utente potrà selezionare il numero di posti, il giorno e l'ora della prenotazione desiderata, per poi concludere l'operazione cliccando sul pulsante "Conferma". Per selezionare la data e l'ora l'utente potrà interagire con delle finestre di dialogo riportate nelle Figure 5.8 e 5.9.

5.1.6 Visualizzazione contatti ristorante

Navigando nell'app sarà possibile visualizzare le informazioni del ristorante, cliccando sulla voce "Informazioni" nel menù di navigazione. Come riportato nella Figura 5.10 saranno visualizzati la posizione, il numero di telefono e l'orario di apertura del ristorante.

Figura 5.4. La figura riporta l'elenco dei prodotti nel menù del ristorante

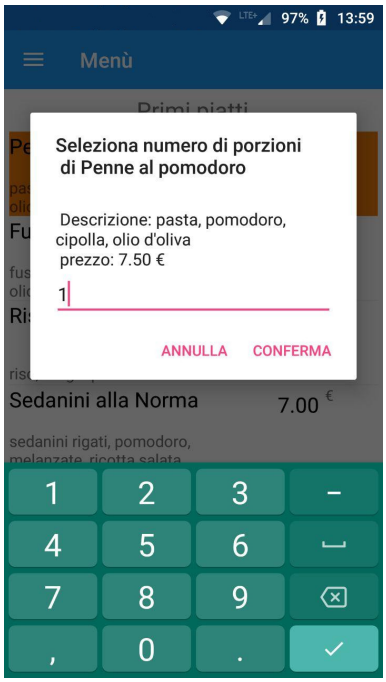

Figura 5.5. La figura riporta la finestra di dialogo, attraverso la quale l'utente potrà selezionare e confermare il numero di porzioni di un prodotto scelto precedentemente

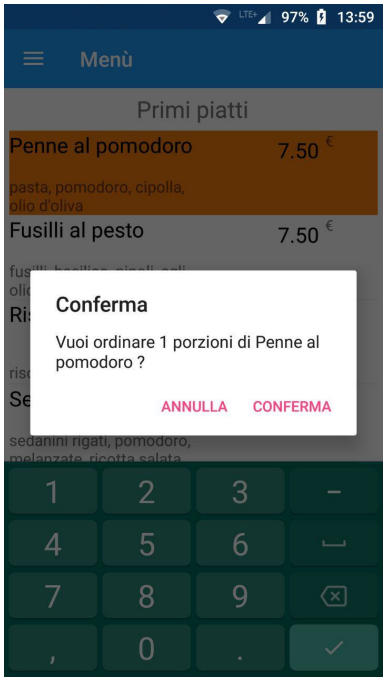

Figura 5.6. La figura riporta la finestra di dialogo attraverso la quale l'utente potrà controllare l'ordine del prodotto selezionato precedentemente

Figura 5.7. La figura riporta la schermata attraverso la quale l'utente può prenotare un tavolo

Figura 5.8. La figura riporta la finestra di dialogo attraverso la quale l'utente potrà selezionare la data della prenotazione del tavolo

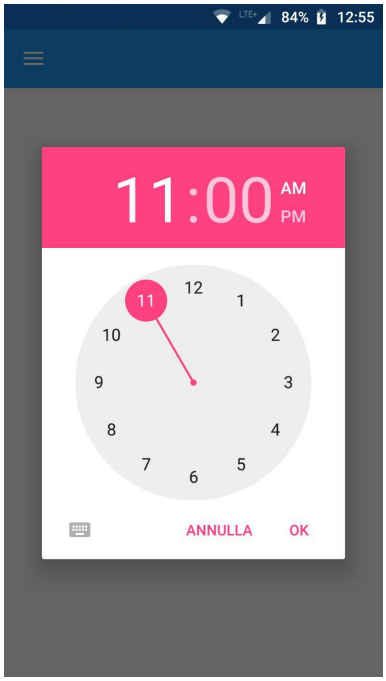

Figura 5.9. La figura riporta la finestra di dialogo attraverso la quale l'utente potrà selezionare l'orario della prenotazione del tavolo

Figura 5.10. La figura riporta la schermata statica in cui sono riportate le informazioni del ristorante

6

Discussione in merito al lavoro svolto

In questo capitolo verranno esposte alcune considerazioni sull'esperienza relativa all'utilizzo di Xamarin. Successivamente verrà presentata la SWOT analysis dell'app; attraverso tale analisi saranno messe in evidenza i punti di forza, di debolezza, le opportunità e le minacce dell'app sviluppata.

6.1 Lezioni apprese sull'utilizzo di Xamarin

Come già discusso nella trattazione di questa tesi, l'universo dello sviluppo mobile è costellato di numerose alternative di cui sicuramente il framework Xamarin (Figura 6.1) è uno dei protagonisti.

Figura 6.1. Logo di Xamarin

Con questa esperienza, abbiamo potuto comprendere quanto Xamarin sia un framework completo e versatile. La solidità di Xamarin è garantita, in primo luogo dal supporto della piattaforma *.NET* e, in secondo luogo, dal contributo della *community* di sviluppatori.

Come già discusso precedentemente, *.NET* fornisce una serie di librerie e linguaggi molto utili per lo sviluppo di app.

Tuttavia, a nostro avvislo, ancora più importante è la semplicità di integrazione di diversi progetti appartenenti a questo ecosistema. Quest'ultima caratteristica semplifica il lavoro e potrebbe abbassare i costi di progetto, qualora fosse necessario

sviluppare progetti su piattaforme di diversa natura (mobile, wearables, app desktop, server o web app), o qualora un'azienda fosse già orientata all'utilizzo di tale framework.

Come già anticipato, Xamarin, nonostante nasca come un framework proprietario, oggi è *open-source*, pur essendo dipendente da software closed-source. Questa scelta porta con sè, in primo luogo, il supporto di una *community* di programmatori che mantiene il framework sempre aggiornato e al passo con le richieste del mercato; in secondo luogo, essa garantisce maggiori possibilità per i programmatori.

Tali aspetti hanno portato Xamarin ad essere sempre più utilizzato sia in ambito didattico che all'interno di aziende; di conseguenza, si è venuta a creare una grande community di sviluppatori. Difatti, consultando quest'ultima, oltre alla documentazione ufficiale, abbiamo avuto la possibilità di recuperare e consultare materiale utile per la risoluzione dei problemi, in cui ci siamo imbattuti in fase di implementazione.

Il progetto descritto nella tesi non è stato il primo in ambito mobile su cui abbiamo lavorato. Avendo già sviluppato app utilizzando *Android native*, avevamo acquisito una serie di abilità e competenze su alcuni aspetti della programmazione mobile. Mi sono dovuto però confrontare con nuovi linguaggi, come *C#* e *XAML*. Ad ogni modo non abbiamo trovato particolarmente difficile l'utilizzo degli stessi, grazie alla loro somiglianza con *C++* e *HTML*, che già conoscevo.

6.2 SWOT analysis

L'analisi SWOT (Strenghts, Weaknesses, Opportunities, Threats) è uno strumento di pianificazione strategica che consente di valutare i punti di forza (Strenghts), di debolezze (Weaknesses), le opportunità (Opportunities) e le minacce (Threats) di un progetto. Gli elementi fondamentali di questo tipo di analisi sono schematizzati nella matrice SWOT riportata nella Figura 6.2.

L'analisi SWOT procede nel seguente modo: per prima cosa si definisce un obiettivo da raggiungere; successivamente si esaminano i quattro aspetti che caratterizzano tale analisi, ovvero:

- *I punti di forza*: sono gli aspetti interni con carattere positivo; rappresentano, quindi, i fattori utili al raggiungimento dell'obiettivo prefissato.
- *I punti di debolezza*: rappresentano gli aspetti di origine interna, ma di carattere negativo; infatti sono i fattori che ostacolano il raggiungimento dell'obiettivo.
- *Le opportunità*: sono condizioni esterne con connotazione positiva, che possono, quindi, aiutare a raggiungere l'obiettivo.
- *Le minacce*: sono le condizioni esterne con connotazione negativa, cioè che possano complicare il raggiungimento dell'obiettivo.

Nelle prossime sottosezioni riporteremo l'analisi SWOT del progetto connesso con la presente tesi.

I punti di forza

Un primo punto di forza è sicuramente apportato al progetto dalla scelta del framework, che permette di avere una grafica accattivante e coerente su dispositivi mo-

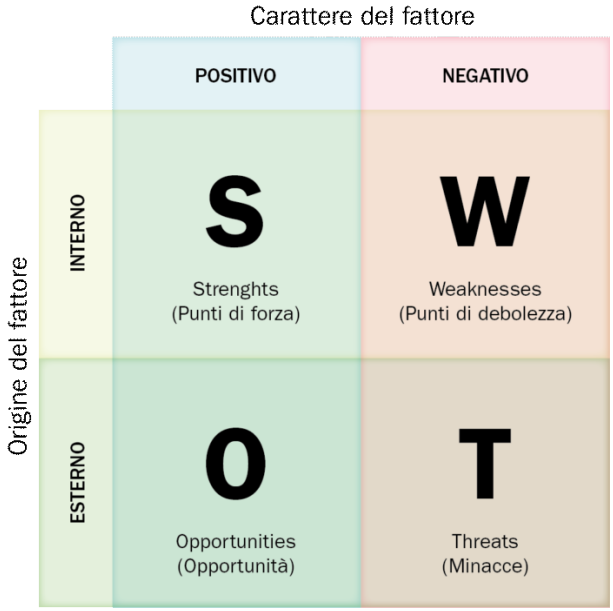

Figura 6.2. I fattori analizzati possono essere distinti per origine del fattore (interno o esterno) e per carattere (positivo o negativo)

bili di diverso tipo, semplificando il lavoro in fase di implementazione e riducendone i costi.

Un secondo vantaggio è determinato dalla semplicità di utilizzo dell'app.

I punti di debolezza

Un primo punto di debolezza potrebbe essere causato da una eventuale mancanza di connessione Internet da parte dell'utente.

Un secondo punto di debolezza potrebbe essere generato dalla scarsa sostenibilità economica del progetto, che potrebbe essere parzialmente sovvenzionato da banner pubblicitari.

Le opportunità

A causa delle condizioni sanitarie attuali, potrebbe essere molto utile l'utilizzo di app come questa, come alternativa all'utilizzo del menù cartaceo.

Le minacce

Una prima minaccia esterna potrebbe essere rappresentata dal grande numero di competitor già presenti sul mercato.

Una seconda minaccia potrebbe essere causata dalla scarsa motivazione, da parte dell'utente, ad installare l'app; l'utente, infatti, potrebbe essere più motivato ad usare servizi online offerti da eventuali competitor che permettono l'utilizzo evitando l'installazione dell'app.

7

Conclusioni e uno sguardo al futuro

In questo capitolo tratteremo le conclusioni in merito al progetto svolto nella presente tesi e, infine, descriveremo alcuni possibili sviluppi futuri.

7.1 Conclusioni

Nella presente tesi è stato utilizzato il framework per lo sviluppo mobile cross-platform Xamarin.

Nella prima parte dell'elaborato abbiamo analizzato tutte le tecnologie disponibili per lo sviluppo mobile. Successivamente, abbiamo descritto il framework Xamarin, sottolineando gli aspetti caratteristici e più innovativi.

Nella seconda parte abbiamo introdotto il progetto oggetto della presente tesi, soffermandoci sull'analisi dei requisiti, nella progettazione e nell'implementazione.

Sono stati descritti, quindi, i requisiti funzionali e non funzionali, per poi passare alla progettazione. In questa seconda fase è stata trattata la struttura generale del progetto, descrivendo inoltre le tecnologie utilizzate per la sua realizzazione.

Durante la fase successiva è stata trattata l'implementazione del progetto, partendo da una panoramica generale dell'IDE. Nel corrispettivo capitolo sono stati, poi, riportati alcuni esempi di codice realizzato per il progetto.

Per completare la trattazione del progetto è stato presentato il manuale utente, utile per valutare l'aspetto e il funzionamento del progetto sviluppato.

Infine, sono state descritte le lezioni apprese sull'utilizzo di Xamarin ed è stata effettuata un'analisi SWOT tramite la quale abbiamo individuato i punti di forza, di debolezza, le opportunità e le minacce dell'app.

7.2 Sviluppi futuri

L'applicazione sviluppata non ha, ancora, tutte le funzionalità necessarie per un suo eventuale rilascio. In primo luogo, l'app realizzata si discosta dal progetto iniziale, in quanto non è stata implementata la parte relativa alla gestione degli ordini da parte del personale del ristorante, nonostante le API possano già potenzialmente

fornire queste funzionalità. In secondo luogo, andrebbe integrata all'interno dell'app la gestione dei pagamenti. Altre funzionalità che potrebbero essere implementate sono la possibilità di visualizzare il menù senza doversi registrare come utente e quella di ordinare direttamente da casa.

8

Ringraziamenti

Vorrei dedicare queste ultime pagine per ringraziare le persone che mi hanno supportato e che hanno creduto in me, perché senza di loro non sarei riuscito ad arrivare dove sono oggi.

Ringrazio in primo luogo la i miei genitori e i miei fratelli, perché mi sono sempre stati accanto è hanno sempre supportato le mie scelte, sopratutto nei momenti di difficoltà. Senza di loro non sarei mai potuto diventare quello che sono e non avrei mai potuto raggiungere i miei traguardi.

Vorrei ringraziare anche il resto dei miei familiari, anche se da lontano sono sempre stati fonte di ispirazione e motivazione e anche se a volte da lontano, li ho sempre percepito come vicini.

Vorrei ringraziare i miei amici di vecchia data che mi sono sempre stati vicini, quelli acquisiti durante il percorso di studi e quelli acquisiti più recentemente. Grazie per essermi stati vicini nei momenti di sfogo e nei momenti di leggerezza.

Infine vorrei esprimere la mia gratitudine al mio relatore il Professore Domenico Ursino per le numerose conoscenze, per la pazienza e l'instancabile passione trasmesse durante la realizzazione di questo progetto.

Riferimenti bibliografici

1. Guida ad un nuovo progetto in Xamarin.Forms https://www.html.it/guide/guida-xamarin/, 2020.
2. Guida ad un nuovo progetto in Xamarin.Forms https://docs.microsoft.com/it-it/visualstudio/get-started/visual-studio-ide?view=vs-2019/, 2020.
3. Previsione della quota di mercato per i sistemi operativi https://www.statista.com/statistics/272307/market-share-forecast-for-smartphone-operating-systems/, 2020.
4. Number of smartphone users worldwide https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/, 2020.
5. what-is-dotnet https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet/, 2020.
6. UI - Xamarin.Forms https://docs.microsoft.com/it-it/xamarin/xamarin-forms/user-interface/, 2020.