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 implementazione di un'app Xamarin per l'organizzazione di party nelle abitazioni nell'era del Covid-19

Design and implementation of an Xamarin app to organize home parties in the Covid-19 era

Relatore

Prof. Domenico Ursino

Candidato

Matteo Ferretti

Anno Accademico 2020-2021

Indice

Elenco delle figure

Elenco dei listati

Introduzione

Negli ultimi anni lo sviluppo tecnologico è stato tale da riuscire a mettere un computer, che fino a poco tempo fa occupava stanze intere, nel palmo di una mano; naquero così gli smartphone.

Nei primi anni, gli smartphone non erano dotati di molte funzionalità, sia perchè era presente un limite tecnologico a livello hardware sia perchè programmare questo tipo di dispositivi era piuttosto complicato.

Tali limiti furono superati in pochissimo tempo; si è passati, quindi, da una situazione iniziale, in cui gli smartphone possedevano poche applicazioni limitate, allo scenario odierno, in cui praticamente esiste un'app per qualsiasi cosa: aggiungere maschere tridimensionali alla fotocamera, impostare promemoria, gestire i dati bancari, etc.

Infatti, questi dispositivi sono ormai paragonabili a un computer di fascia medio-alta, e nei prossimi anni si assisterà sicuramente a uno sviluppo sempre più intenso, che permetterà di realizzare operazioni complesse che hanno bisogno di una grande potenza computazionale su dispositivi con dimensioni estremamente ridotte.

In questo sviluppo incredibile è nato e si è sviluppato un nuovo tipo di mercato, ovvero quello delle app, che consiste nell'offrire sempre nuove applicazioni che vanno a soddisfare le esigenze dell'utente, cercando di puntare ad una porzione ben determinata del mercato, cercando sempre l'idea originale e diversa dalla massa.

La nostra applicazione vuole fare proprio questo, ovvero inserirsi nell'ambito del mobile, andando, però, a sviluppare idee originali grazie anche al periodo storico in cui ci troviamo.

Infatti, in questo momento, vi è una pandemia globale dovuta al virus Covid-19. Esso si è manifestato inizialmente a Wuhan in Cina nel 2019 per poi diffondersi in tutto il mondo.

Il virus attualmente è ancora in circolazione, ma dopo una prima fase di osservazione e studio del virus, il mondo intero è passato al contrattacco, grazie a ferrei lockdown di massa durati mesi e ai vaccini finalmente in distribuzione.

Questo progetto è finalizzato alla realizzazione di un'applicazione per la gestione di eventi domestici, cene di compleanno, riunioni con amici e parenti nell'era del Covid-19.

Attraverso l'utilizzo della nostra app non solo sarà più semplice e veloce per l'organizzatore procedere alla distribuzione degli inviti ai vari ospiti e, conseguente-

mente, conoscere il numero esatto delle persone che parteciperanno, ma, nel creare il menù da offrire durante l'evento, si avrà sempre la possibilità di controllare se alcuni invitati presentano particolari intolleranze e, quindi, sulla base di esse, modulare lo stesso.

Altra caratteristica della nostra app è quella di tenere traccia di tutti gli eventi già svolti, con indicazione delle persone e dei menù realizzati in ogni singola occasione, al fine di evitare di riproporre a breve distanza i medesimi piatti a identici commensali.

Data la situazione attuale, abbiamo pensato di implementare una serie di accorgimenti che vanno ad aiutare tutti gli utenti nella battaglia contro il Covid-19 grazie alla prevenzione. Infatti, nella nostra app, abbiamo inserito le norme igieniche, le norme per la ristorazione e un link al sito del governo in modo da essere aggiornati ogni volta che si vuole organizzare un evento sulle norme attuali.

Abbiamo, inoltre, fatto in modo di poter dichiarare di essere idonei per un'eventuale festa tramite un'autocertificazione che va compilata prima di accettare l'invito. L'utente può, inoltre dichiararsi "malato" tramite un pulsante che avverte l'organizzatore e che, quindi, gli impedisce di invitarlo alla festa.

L'organizzatore, oltre allo stato dell'utente, può invitare gli altri utenti solo se è in zona gialla o arancione, e, in base al colore della loro regione, gli vengono permessi l'invito e la creazione dell'evento; infatti in zona rossa viene disabilitata la possibilità di creare eventi.

La tesi è strutturata come di seguito specificato:

- Nel Capitolo 1 si parlerà della pandemia globale dovuta al virus Covid-19, della sua origine, del suo sviluppo e di come ha reagito il mondo, ma soprattuto l'Italia, trovandosi a fronteggiare una situazione delicata come quella che stiamo vivendo.
- Nel Capitolo 2, dopo aver presentato brevemente le diverse tipologie di app attualmente presenti, si illustreranno i pro e i contro delle app ibride, indicando le motivazioni che hanno spinto alla scelta di tale tipologia di app per la realizzazione del presente progetto. Vedremo, poi, la storia di Xamarin, come è stato concepito, cosa è diventato, e quali sono le sue potenzialità.
- Nel Capitolo 3, verrà trattato, innanzitutto, l'analisi dei requisiti funzionali e non, per poi concentrarsi sulla fase di progettazione, in cui vengono presentati i diagrammi UML, per comprendere il funzionamento dell'applicazione, e i mockup, per avere un'idea del suo aspetto.
- Nel Capitolo 4 verrà mostrata l'implementazione del progetto, analizzando i listati di codice.
- Il Capitolo 5 è il manuale utente. In esso vengono illustrati tutti i passaggi che un utente generico deve effettuare per poter utilizzare l'applicazione, illustrandone le funzionalità principali.
- Nel Capitolo 6 verrà presentata una discussione del lavoro svolto, illustrando la SWOT Analysis, in modo da poter comprendere i punti di forza e di debolezza dell'app realizzata, con un'analisi delle app concorrenti nelle stessa ipotetica fascia di mercato in cui si collocherebbe la nostra applicazione.
- Infine, il Capitolo 7 riporta le conclusioni del progetto realizzato, indicando possibili miglioramenti e sviluppi futuri.

1

Il Covid-19 e i protocolli di sicurezza

1.1 Il Covid-19

La pandemia di COVID-19, attualmente in corso, riguarda la diffusione a livello globale della cosiddetta "malattia da nuovo coronavirus", meglio nota con la sigla di COVID-19.

Il COVID-19, acronimo dell'inglese **CO**rona **VI**rus **D**isease **19**, è una malattia infettiva respiratoria causata dal virus denominato SARS-CoV-2 appartenente alla famiglia dei coronavirus [10].

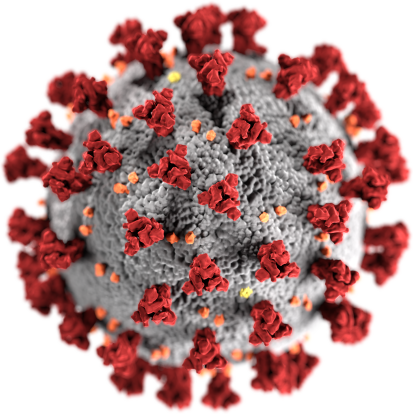


Figura 1.1: Modello 3D del Covid-19

I primi casi conosciuti coinvolsero principalmente lavoratori del mercato umido di Wuhan, in Cina, in cui si vendevano pesce e altri animali, anche vivi.

Nelle prime settimane di gennaio 2020 gli scienziati individuarono in questi soggetti strane polmoniti causate da un nuovo coronavirus, designato SARS-CoV-2 (sindrome respiratoria acuta grave da coronavirus 2) [7], risultato essere simile almeno al 70% della sua sequenza genica a quella del SARS-CoV.

Alla fine del mese di gennaio 2020 non erano ancora state ben determinate le caratteristiche del virus, sebbene fosse accertata la capacità di trasmettersi da persona a persona, e permanevano incertezze sulle esatte modalità di trasmissione e sulla patogenicità (la capacità di creare danno).

Gli ammalati accusano sintomi simili all'influenza come dermatiti, febbre, tosse secca, stanchezza, difficoltà di respiro. Nei casi piu gravi, spesso riscontrati in soggetti già gravati da precedenti patologie, si sviluppa polmonite, insufficienza renale acuta fino ad arrivare anche al decesso.

1.2 I protocolli di sicurezza attuati dal governo Italiano

Inizia il primo lockdown di massa della storia. 60 milioni di persone appartenenti alla provincia di Hubei, di cui 11 nella sola città di Wuhan, entrano in un rigido lockdown. Strade deserte e servizi ridotti al minimo.

Un uomo residente a Codogno risulta positivo al coronavirus: è il paziente 1 in Italia. Nel giro di poche ore vengono registrate le positività di altre quattordici persone [9].

1.2.1 Fase 1

Il 23 febbraio scatta l'implementazione delle "zone rosse" in 11 comuni tra Lombardia e Veneto, tra cui Codogno e Vo' Euganeo (Figura 1.2). Viene istituito il divieto di accesso o di allontanamento dal territorio comunale e vengono sospese manifestazioni, eventi e ogni forma di riunione in luogo pubblico o privato, anche di carattere culturale.

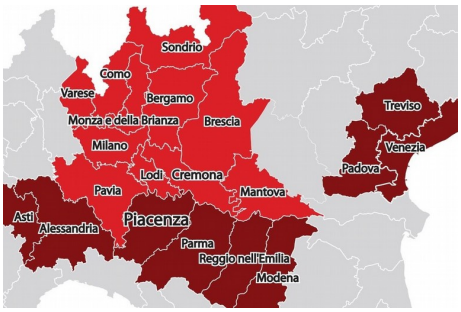


Figura 1.2: Prime zone rosse in Lombardia e Veneto

Il 4 marzo l'Italia annuncia la sospensione delle attività scolastiche in tutta la nazione, estendendo a tutto il Paese le misure già in vigore nelle regioni del nord; l'intero Paese è ora in lockdown, primo tra gli stati occidentali ad adottare misure così severe e restrittive [9].

1.2.2 Fase 2

Il 26 aprile il Presidente del Consiglio Giuseppe Conte annuncia le misure per il contenimento dell'emergenza Covid-19 nella cosiddetta "fase due", al via il 4 maggio dopo più di 2 mesi di lockdown (Figura 1.3). Le misure prevedono il ritorno al lavoro di 4 milioni di italiani e consentono visite ai familiari nella stessa regione giustificando lo spostamento tramite un'autocertificazione resa disponibile dal governo.

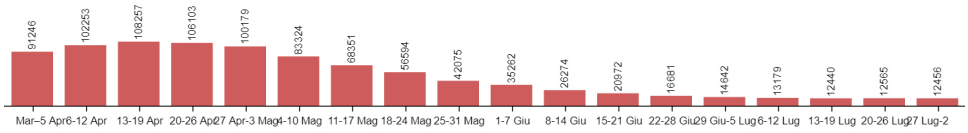


Figura 1.3: Diminuzione della curva dei contagi dopo le restrizioni di inizio marzo

Il 18 maggio l'Italia comincia una nuova fase di riaperture che segna, di fatto, la fine del lockdown cominciato a marzo. Bar e ristoranti riaprono, così come molte filiere produttive.

È possibile incontrare persone al di fuori del proprio nucleo familiare o affettivo e, per spostarsi all'interno della stessa regione, non è più necessaria l'autocertificazione. Bisogna trovarsi sempre a disanza di almeno 1 metro, e con mascherina, in alcune regioni da indossare anche all'aperto (Figura 1.4).

Il Dpcm dell'11 giugno, che entrerà in vigore il 15 giugno, prevede tutta una serie di aperture e alleggerimenti rispetto alle settimane precedenti. Riaprono le aree giochi e i centri estivi anche per i bambini da zero a tre anni, oltre alle sale scommesse. Via libera, inoltre, a cinema e teatri e agli spettacoli all'aperto, per un massimo, rispettivamente, di duecento e mille spettatori. Decade l'obbligo di mascherina all'aperto, anche se in alcune regioni (come la Lombardia) questa decisione viene prorogata alle settimane successive.

Il 14 settembre in Italia riaprono finalmente anche le scuole [9].

Figura 1.4: Immagini di vita quotidiana in Fase 2

1.2.3 Fase 3

Ma nel mese di ottobre in Italia esplode la seconda ondata. Dopo un settembre con i contagi più o meno sotto controllo, la curva esplode e lascia presagire la necessità di nuove misure di sicurezza. Inizialmente arriva un nuovo Dpcm in cui sono previste nuove misure restrittive per bar e ristoranti, sport di contatto, scuola e didattica a distanza con anche la chiusura di palestre e piscine.

Il 4 novembre un nuovo Dpcm firmato dal Presidente del Consiglio divide l'Italia in 3 zone (Figura 1.5) con diverse restrizioni che entrano in vigore il 6 novembre.

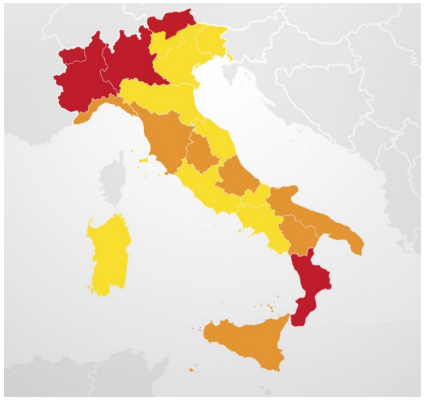


Figura 1.5: Divisione dell'Italia

La suddivisione prevede una fascia gialla, una arancione e una rossa, in base alla gravità della situazione.

Nella zona rossa, quella più a rischio, scatta, di fatto, un lockdown come il precedente: non si può uscire di casa se non per motivi di prima necessità (Figura 1.6).

In zona arancione si può uscire di casa ma non dal proprio comune se non per questioni di salute, studio o lavoro.

In zona gialla ci si può spostare liberamente all'interno della regione e tra regioni esclusivamente di colore giallo; non si può, infatti, andare in una regione di colore arancione o rosso [9].

Figura 1.6: Regole associate alle aree gialle, arancioni e rosse

2

Xamarin

Con l'avvento degli smartphone è aumentato il bisogno di strumenti con cui creare applicazioni software che fossero in grado di funzionare su numerosi dispositivi, anche con sistemi operativi diversi. Per questo, nel mondo della programmazione mobile, esistono diversi modi per creare un'app, ciascuno con i suoi vantaggi e svantaggi.

2.1 Tipologie di app

Un'app mobile, o semplicemente app, è un programma progettato per poter funzionare su dispositivi portabili, quali smartphone, tablet o smartwatch. Un'app può appartenere a una delle seguenti categorie (Figura 2.1):

- app native;
- web app;
- app ibride;

Figura 2.1: Le diverse tipologie di app

2.1.1 Le app native

Le app native sono app create per funzionare specificatamente su un sistema operativo mobile. I sistemi operativi più diffusi sono iOS e Android (Figura 2.2). Attualmente il sistema operativo più diffuso è Android (che copre circa il 62,94% del mercato contro il 33,9% di iOS [13]); questo perchè gli smartphone dotati di questo sistema operativo risultano essere più economici rispetto a quelli basati sul sistema operativo iOS.

Figura 2.2: iOS e Android

I vantaggi principali delle app native sono i seguenti:

- hanno maggiore velocità, affdabilità e reattività, che si traduce in una migliore esperienza utente;
- accedono più facilmente ed efficientemente alle funzionalità del telefono (come fotocamera, microfono, bluetooth, etc.);
- possono utilizzare le notifiche push;
- non necessitano di una connessione Internet, quindi possono funzionare anche in modalità offline.

Quindi, come si evince dall'elenco, le app native sono in grado di sfruttare al massimo le varie funzionalità del dispositivo su cui sono installate, garantendo un'esperienza utente ottimale. Lo svantaggio principale delle app native è la portabilità, in quanto è necessario creare una versione nativa dell'app per ogni sistema operativo in commercio.

Infatti, dal tipo di dispositivo dipende il linguaggio di programmazione e, quindi, il modo con cui vengono sviluppate le app native; per iOS, ad esempio, si ricorre in genere a Object C, XCode e Swift, mentre per Android a Java o Kotlin.

2.1.2 Le web app

Le web app non sono altro che la versione mobile di un sito web, accessibile tramite un qualsiasi browser installato sul dispositivo. La realizzazione di una web app viene effettuata tramite gli strumenti di sviluppo classici dei siti web, cioè con linguaggi di mark-up (HTML5 e CSS3) e linguaggi di programmazione lato client (Javascript) e lato server (PHP). Le web app sono all'estremo opposto rispetto alle app native; esse presentano numerosi vantaggi da non sottovalutare.

In particolare:

- funzionano come un sito web; di conseguenza, si comportano su tutti i dispositivi allo stesso modo, senza alcuna differenza a livello di codice (Figura 2.3);
- non è necessario installarle, quindi non consumano la memoria del dispositivo;
- consumano un numero di risorse notevolmente inferiore rispetto a un'app nativa o ibrida.

Figura 2.3: Esempio di una web app

Le web app sembrerebbero più vantaggiose delle app native. In realtà, esse presentano numerosi svantaggi; in particolare:

- non possono sfruttare al massimo le funzionalità del dispositivo, quindi sono meno performanti e più lente;
- in genere non funzionano in assenza di una connessione ad Internet;
- non vengono scaricate, quindi si ha una bassa fidelizzazione.

È da tener conto però, che gli obiettivi di una web app sono diversi da quelli di un'app nativa. Le web app hanno lo scopo di rendere disponibile un contenuto al maggior numero possibile di utenti, mentre le app native hanno lo scopo di fornire servizi sfruttando al massimo le capacità dello smartphone.

2.1.3 Le app ibride

Le app ibride sono una via di mezzo fra le web app e le app native; esse, infatti, sono app in grado di funzionare su sistemi operativi diversi (Figura 2.4) utilizzando lo stesso codice sorgente (o almeno una buona parte).

Le app ibride si compongono di una parte cross-platform (cioè una parte in comune per tutti i diversi sistemi operativi) e di una parte specializzata (specifica per il sistema operativo su cui devono funzionare).

Questo risulta essere un enorme vantaggio perchè si riesce ad ottenere la portabilità tipica delle web app avendo le performance di un'app nativa. Il linguaggio di programmazione più usato per lo sviluppo di app ibride è C# (C-Sharp).

Rispetto alla app native, quelle ibride sono meno costose e più veloci da sviluppare. Inoltre, poichè esiste solo una versione dell'app gli aggiornamenti e l'aggiunta di nuove feature, le app ibride sono più semplici ed efficaci; infatti, negli ultimi anni, è stata registrata una crescente adozione della soluzione cross-platform, come si evince da un recente sondaggio Forrester [16] che ha rilevato che due terzi degli sviluppatori scelgono un approccio multipiattaforma o basato sul Web rispetto agli strumenti nativi.

Le app ibride, però, risultano essere più lente delle app native; questo perchè tali app si devono poter adattare ad una vasta gamma di sistemi operativi e dispositivi differenti.

Figura 2.4: App ibride multipiattaforma

2.2 Perchè scegliere le app ibride

Riassumiamo, allora, i vantaggi e gli svantaggi che abbiamo usato come criterio per lo sviluppo della nostra applicazione come app ibrida.

2.2.1 Vantaggi

- *Si ha un unico codice per tutte le piattaforme:* solo una minima parte di codice dovrà essere specializzata per i diversi sistemi operativi.
- *Le app ibride sono molto più performanti che in passato:* infatti, il divario delle performance fra app ibride e app native si è ridotto notevolmente negli ultimi anni. Le soluzioni moderne per lo sviluppo di app ibride offrono la stessa accelerazione delle prestazioni basata su hardware delle app native. Ad esempio, Ionic [8] afferma di avere sviluppato un framework che permette di realizzare applicazioni con le stesse performance di un'app nativa (a parità di funzionalità).
- *Aggiornamento rapido dell'app:* viene generata una sola versione, indipendentemente dal numero di piattaforme. Ciò significa che se si vogliono aggiungere delle nuove funzionalità oppure si vuole effettuare un aggiornamento, basterà modificare (il più delle volte) una porzione di codice condiviso.

2.2.2 Svantaggi

- *Performance e sovraccarico su dispositivi meno performanti:* l'utilizzo di un'app ibrida, piuttosto che un'app nativa, può introdurre un maggior carico di lavoro per i dispositivi mobili, anche se, negli ultimi anni, la realizzazione di API molto performanti e il miglioramento delle caratteristiche tecniche dei dispositivi mobili hanno reso il gap con le app native meno sensibile. C'è comunque da evidenziare, per onestà intellettuale che nella realizzazione di app molto dinamiche come giochi, realtà virtuale, contenuti multimediali o elaborazioni grafiche coinvolgenti, le app native sono ancora in netto vantaggio.
- *Componenti e plugin:* soluzioni multipiattaforma ibride moderne sono in grado di accedere a quasi tutte le funzionalità native di un dispositivo, come fotocamera, microfono, gps, giroscopio, etc. Tuttavia, accedere a queste funzionalità spesso comporta l'utilizzo di plugin il che aggiunge complessità e dipendenze allo sviluppo; inoltre, la necessità di ricorrere a codici diversi e di terze parti comporta un minor controllo sulla qualità e sulla flessibilità.
- *Dipendenze dai frameworks di terze parti:* i frameworks per lo sviluppo di app ibride (Fig 2.5) sono sviluppati da società diverse, e non dagli stessi sviluppatori dei sistemi operativi. Scegliere un approccio multipiattaforma significa scommettere negli sviluppatori e nella durata e stabilità del relativo framework (Xamarin, React Native, Ionic, Appcelerator, etc.). Infatti, potrebbe accadere che tali progetti vengano abbandonati e non più supportati.

Figura 2.5: Alcuni dei framework per la programmazione di app

2.3 Xamarin

La storia di Xamarin nasce nell'ormai lontano 1999, quando Nat Friedman [2] e Miguel de Icaza [1], entrambi sviluppatori, fondano la Helix Code, che cambierà poi nome in Ximian (Figura 2.6a) nel 2001.

L'obiettivo era quello di dare vita ad un software gratuito per la piattaforma Linux. Negli anni successivi la società focalizzerà le sue risorse per portare avanti il progetto Mono, che acquisirà consistenza quando l'azienda sarà acquistata da Novell, la quale finanzierà lo sviluppo delle due versioni per Android e iPhone, ovvero Monodroid e Monotouch.

Mono (Figura 2.6b) è una implementazione cross-platform open source del framework .NET. Questa include il Common Language Runtime(CLR) cioè l'ambiente di esecuzione del Common Intermediate Language(CIL). Quest'ultimo rappresenta il linguaggio intermedio in cui i compilatori della piattaforma .NET traducono i linguaggi ad alto livello, come C#. Trattandosi di una macchina virtuale, la Mono CLR è stata sviluppata per diverse piattaforme, come Android, iOS, OS X e per i sistemi basati su Windows [5] .

Nel 2011, dopo l'acquisizione dell'azienda Ximian da parte di Microsoft, nasce finalmente Xamarin (Figura 2.6c). 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 dell'omonimo framework Xamarin [3].

(a) Logo di Ximian (b) Logo di Mono (c) Logo di Xamarin (d) Logo di C#

Figura 2.6: Loghi delle varie aziende

2.3.1 Xamarin Framework

L'obiettivo di Xamarin è quello di offrire strumenti che permettano agli sviluppatori di raggiungere i principali sistemi operativi per dispositivi mobili, ovvero Android, iOS e Windows, massimizzando il riutilizzo del codice. Per fare questo, come già detto, Xamarin sfrutta Mono, il quale permette di utilizzare C# (Figura 2.6d), il versatile linguaggio di programmazione orientato agli oggetti di Microsoft, per sviluppare applicazioni multipiattaforma. Mono e C# sono, quindi, la base su cui Xamarin costruisce i propri strumenti e le proprie tecnologie di sviluppo.

Questo è un fattore fondamentale per gli sviluppatori .NET, che possono, quindi, riutilizzare le proprie competenze anche su sistemi non Microsoft.

Per rendere possibile tutto questo, Xamarin ha sviluppato delle librerie che rielaborano ed espongono in forma .NET/Mono le API native delle varie piattaforme.

Tali librerie sono:

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

Oltre ad esse si aggiunge Xamarin.WinPhone che, ovviamente, è stata sviluppata facilmente, dal momento che, per lo sviluppo di applicazioni per Windows Phone, viene fatto già uso nativamente di C# e di API in forma .NET (anche se ormai il mercato dei Windows phone è praticamente azzerato). Con un unico codice condiviso, basato sul linguaggio C#, in ottica totalmente orientata a .NET, gli sviluppatori possono usare gli strumenti Xamarin per scrivere applicazioni native Android e iOS, oltre che, ovviamente, Windows, con interfacce utente native. Da ciò è facile notare, tuttavia, come l'eventuale ampliamento futuro del supporto verso altri sistemi operativi possa essere effettuato in maniera più o meno rapida aggiungendo esclusivamente una nuova libreria *Xamarin.X*. Oltre al runtime e alle librerie, sono stati pensati e creati appositamente per questo scopo degli ambienti di sviluppo e dei tool (che sono stati rilasciati come progetti open source dal 2016, a cui la comunità di sviluppatori può contribuire). In realtà, Xamarin offre agli sviluppatori tutto ciò di cui hanno bisogno per sviluppare, pubblicare, analizzare e mantenere app native per Android, iOS e Windows (Figura 2.7).

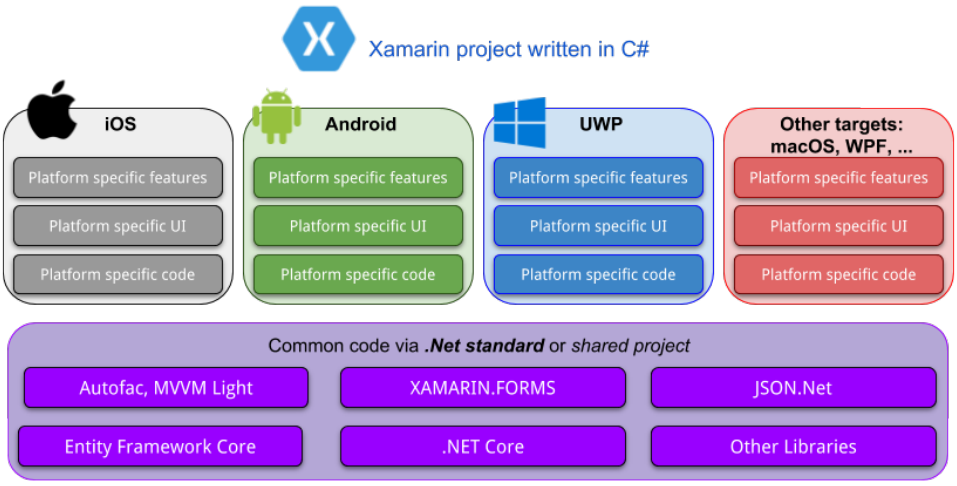


Figura 2.7: Xamarin Framework

2.3.2 Condivisione del codice

Come riportato nella documentazione ufficiale online di Xamarin, C# è considerato un linguaggio di programmazione ottimale per lo sviluppo di applicazioni mobile.

Qualsiasi cosa si possa fare in Objective-C, Swift o Java, può essere fatta in C#. Xamarin ha scelto, dunque, C# come unico linguaggio, permettendo agli sviluppatori di ottenere applicazioni ibride senza dover necessariamente conoscere altri linguaggi, e utilizzando, come vedremo in seguito, un unico ambiente di sviluppo. Tutto ciò è possibile grazie alle sopracitate librerie sviluppate da Xamarin, le quali permettono di accedere a qualsiasi API dei sistemi operativi nativi.

2.3.3 Architettura di una soluzione

In passato Xamarin permetteva di realizzare attraverso le sue librerie un layer comune condiviso tra le varie piattaforme, scritto in C#, che rappresenta il back-end, ovvero la logica dell'app *code-behind*. Esso, a seconda della complessità del codice dell'applicazione da realizzare, può coprire in media il 75% del codice nativo, che può essere condiviso (in alcuni casi questa percentuale raggiunge anche il 100%).

Su questa base comune di back-end si poggiava, poi, la realizzazione dell'interfaccia grafica nativa (Figura 2.8), una per ogni piattaforma considerata, sfruttando i linguaggi di markup e i tool nativi. Nello specifico, quindi, si aveva:

- Android: C# + XML;
- iOS: C# + XIB/Storyboard;
- Windows Phone: C# + XAML.

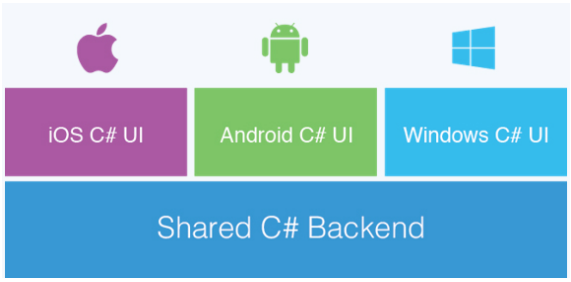


Figura 2.8: Vecchio approccio di Xamarin

Nel maggio del 2014, viene introdotta la nuova libreria *Xamarin.Forms* (Figura 2.9), che permette di realizzare la condivisione del codice tra le piattaforme a tutti i livelli, compreso quello dell'interfaccia grafica, massimizzando, in questo modo, il riutilizzo del codice. L'obiettivo a cui si vuole ambire, infatti, è il raggiungimento del 100% del codice condiviso. A tal fine, *Xamarin.Forms* fornisce un livello di astrazione che consente di definire l'interfaccia grafica una solta volta (Figura 2.10), attraverso una derivazione di XAML (eXtensible Application Markup Language), oppure da codice C#.

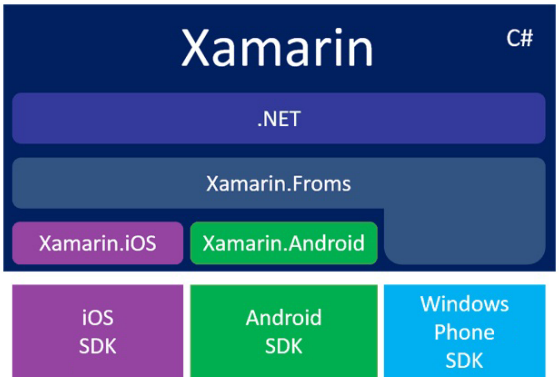


Figura 2.9: La nuova libreria *Xamarin.Forms*

Lo sviluppatore utilizzerà esclusivamente tali astrazioni per costruire l'interfaccia grafica della propria applicazione attraverso i classici componenti (pagine, layout, viste, pulsanti, etichette, etc.). Solo in fase di esecuzione, *Xamarin.Forms* effettuerà il rendering selezionando i corrispettivi elementi visuali appropriati, specifici del sistema su cui l'applicazione sta girando, ottenendo un aspetto nativo.

Quando si aggiunge una Label, Xamarin trasformerà questa in una UILabel per iOS, un TextView per Android e una TextBlock per Windows Phone.

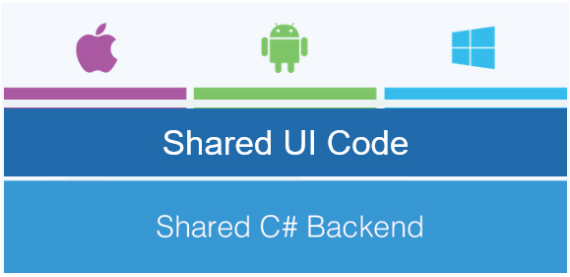


Figura 2.10: Nuovo approccio di Xamarin

All'interno del progetto condiviso, vi è il codice che può funzionare con certezza su tutte le piattaforme. Ma ci sono, tuttavia, degli aspetti di cui tenere conto. Il primo è che *Xamarin.Forms* è in grado di mappare solo quegli elementi che hanno una corrispondenza in tutte le piattaforme. Ad esempio, sia iOS che Android che Windows hanno caselle di testo ed etichette, per cui *Xamarin.Forms* definisce controlli Entry e Label che, con certezza, saranno utilizzabili; tuttavia, iOS, Android e Windows gestiscono in modo diverso i due elementi, con proprietà e comportamenti differenti. *Xamarin.Forms*, quindi, definisce controlli che hanno esclusivamente proprietà e comportamenti comuni a tutti, lasciando fuori aspetti specifici di ciascuna piattaforma (anche se recentemente è stata inserita la possibilità di utilizzare controlli nativi).

Funzionalità come l'accesso ai sensori, ad esempio, essendo basate su API completamente diverse tra loro, necessitano di essere gestite in modo differente a seconda della piattaforma.

In *Xamarin.Forms* ci sono tre possibilità per fare ciò, ovvero:

- *Custom Renderer:* riguarda la personalizzazione di tutti i layout e gli elementi visuali in generale, utilizzando controlli nativi in modo diretto, tramite l'implementazione di una classe per ogni progetto specifico, per aggiungere proprietà tipiche di quella piattaforma;
- *Dependency Injection:* riguarda l'accesso a funzionalità hardware e software (sensori, fotocamera, file system) specifiche della piattaforma, realizzato tramite la definizione di un'interfaccia comune per descrivere i membri di una classe, e fornendo nei progetti specifici la relativa implementazione;
- *Effects:* riguarda la situazione in cui si vogliono utilizzare controlli nativi, ma senza dover intervenire anche sul loro comportamento, ad esempio nella modifica degli stili.

Il secondo aspetto è che *Xamarin.Forms* è una piattaforma piuttosto giovane e in piena evoluzione; quindi, non sempre è disponibile ciò che siamo abituati ad utilizzare nelle piattaforme specifiche o in altri contesti di sviluppo. C'è, tuttavia, da sottolineare che la sua evoluzione è talmente rapida che le nuove versioni includono sempre importanti miglioramenti, strumenti e nuovi controlli. Le funzionalità di *Xamarin.Forms* sono già presenti alla creazione di un nuovo progetto attraverso l'omonimo pacchetto NuGet (il package manager gratuito realizzato per le piattaforme Microsoft), dal quale è possibile, anche, scaricare numerosi plugin per Xamarin, ovvero librerie per l'accesso a funzionalità specifiche della piattaforma, tramite API unificate fruibili dai progetti condivisi, senza ricorrere alle tecniche descritte sopra.

Nella Figura 2.11 è riportato il classico esempio di un'architettura cross-platforms che offre il riutilizzo del codice attraverso le varie piattaforme.

Possiamo, quindi, dividere la nostra soluzione in due macro progetti, ovvero:

- *Core Project:* è il cuore della nostra applicazione e si compone degli elementi essenziali, come il **Data Access Layer**, che esporrà un set di API per l'accesso ai dati del database. Tali API che saranno richiamate dal Business Layer tramite l'uso di opportune interfacce e, appunto, questo *Business Layer*, che conterrà tutta la logica e le operazioni principali volte a garantire l'interazione con l'utente.
- *Platform-Specific Application Projects:* è l'insieme dei progetti delle piattaforme che si desidera supportare per la propria app.
  In particolare, ogni progetto dovrebbe essere costituito dalla gestione dell'interfaccia utente (User Interface - UI) e dalla gestione di specifiche funzionalità delle piattaforme tra il layer relativo alla logica di business della nostra applicazione e l'UI.

Referendoci alla Figura 2.11, possiamo identificare il Core Project con il contenitore *Share Code*, che permette una condivisione del codice attraverso PCL o SAP, e le piattaforme che si desidera supportare, che rientrano nei Platform-Specific Application Project.

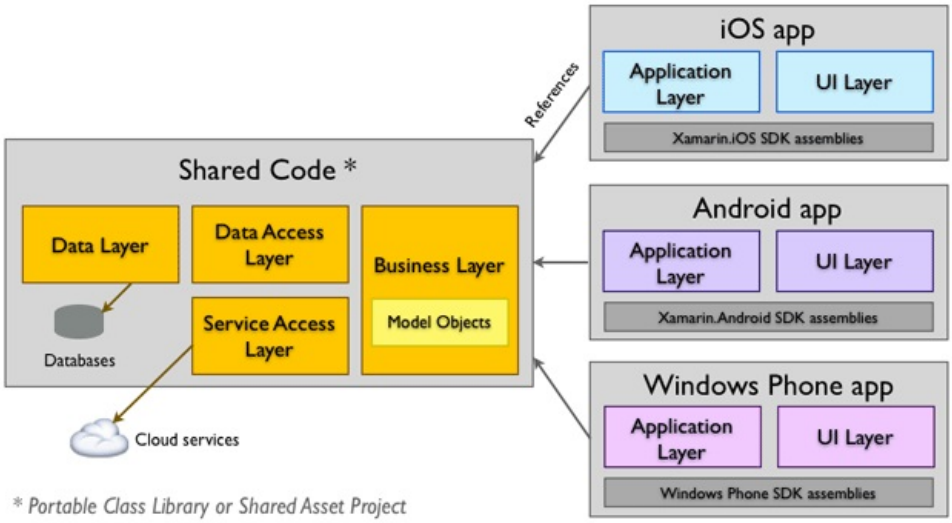


Figura 2.11: Architettura di un'app cross-platform

Quindi, Xamarin offre due diversi approcci per la creazione di soluzioni cross-platform; questi verranno presi in esame nelle prossime sottosezioni:

Architettura SAP (Shared Assets Project)

Tale approccio offre la possibilità di condividere il codice attraverso l'intera soluzione, sfruttando le direttive del compilatore per includere differenti porzioni di codice, specifiche per ogni piattaforma. I progetti basati su questo approccio hanno un limite, ovvero i progetti Android e iOS accedono ad una versione del .NET framework che differisce da quella usata per i progetti Windows. È necessario differenziare il codice per ogni piattaforma supportata per effettuare l'operazione di interesse, come, ad esempio, l'accesso al database.

Questo problema viene risolto introducendo delle direttive e dei simboli C# per il compilatore. In particolare:

- Le direttive sono le seguenti:
  - `#if`
  - `#elif`
  - `#endif`
- I simboli sono i seguenti:
  - `__MOBILE__` : impiegato nei progetti iOS e Droid;
  - `__IOS__` : definito nel progetto iOS;
  - `__ANDROID__` : definito nel progetto Droid;
  - `__ANDROID_nn__` : definito per Android; non rappresenta il livello delle API per cui eseguire una particolare azione;
  - `WINDOWS_PHONE_APP` : definito nel progetto per Windows Phone;
  - `WINDOWS_UWP` : utilizzato nel progetto delle UWP (Universal Windows Platform).

Tramite la combinazione dei simboli e delle condizioni `#if-#elif`, possiamo integrare funzionalità specifiche delle piattaforme di interesse, aggiungendo, nei metodi che lo richiedono, un blocco di istruzioni condizionali. Così facendo, durante la compilazione, lo Shared Project sarà compilato come parte di ogni progetto e, attraverso le direttive del compilatore, verrà utilizzato solo il codice specifico per quella piattaforma.

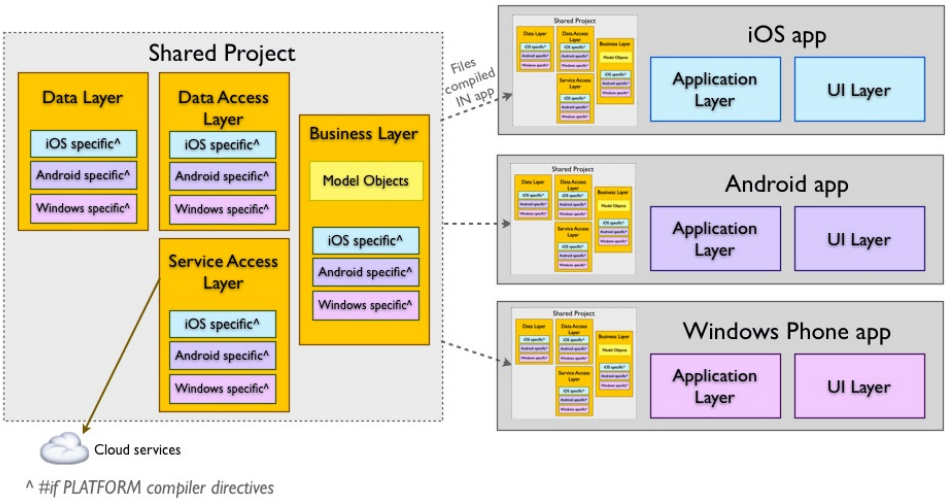


Figura 2.12: Architettura dell'app secondo l'approccio Shared Project

Architettura PCLs (Portable Class Libraries)

L'approccio PCLs invece, a differenza degli Shared Project, permette di usare un sottoinsieme di librerie offerte dal .NET framework per sfruttare lo stesso codice nei progetti che compongono l'intera soluzione. L'insieme di librerie da utilizzare è limitato dalle piattaforme che si desidera supportare per la propria applicazione.

Nella Figura 2.13 le frecce servono a mostrare che ogni "Platform-Specific Application Project" (iOS app, Android app, Windows Phone app) referenzia la PCL. Esse, inoltre mostrano che è possibile impiegare caratteristiche di una determinata piattaforma attraverso l'utilizzo del design pattern Dependency Injection (DI). Questo offre le linee guida per includere le funzionalità specifiche di una piattaforma nella nostra soluzione, fornendo un modo per implementare i pattern Strategy e/o Bridge descritti da Gang of Four (GoF).

Applicando le DI definiamo nel codice condiviso interfacce (classi astratte) che vengono implementate (estese) in ogni piattaforma tramite sottoclassi. A questo punto, sarà possibile integrare tali implementazioni specifiche all'interno della PCL.

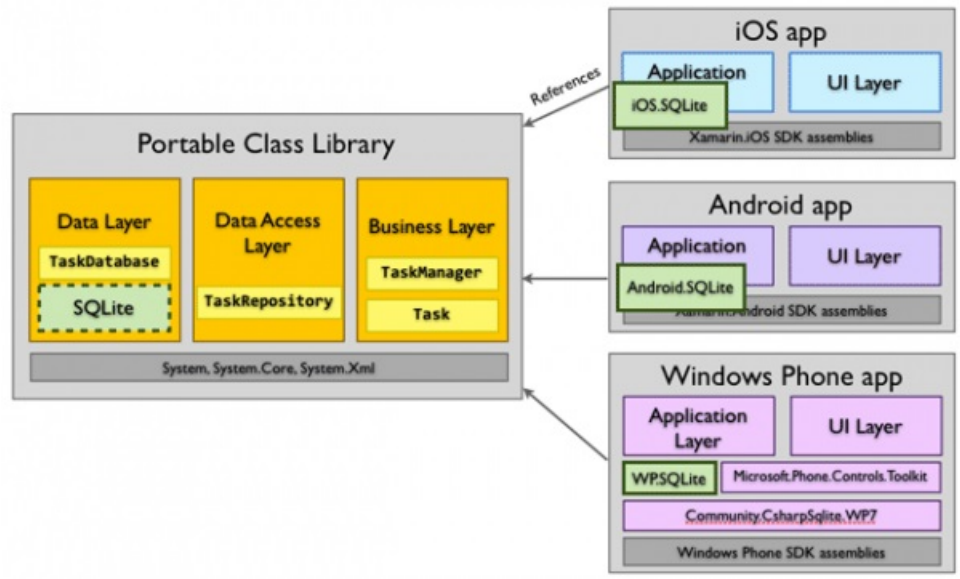


Figura 2.13: Architettura di un'app cross-platform basata su PCLs

2.3.4 Ambiente di sviluppo - Visual Studio 2019

Un altro aspetto fondamentale per lo sviluppo di applicazioni tramite Xamarin è il sistema operativo su cui sviluppare. Infatti Xamarin limita la scelta solo a Mac OS X e Windows , escludendo, almeno per ora, gli utenti Linux. Xamarin offre la possibilità di usare due diversi IDE per lo sviluppo di app native, cross platform e desktop, ovvero:

- Xamarin Studio: realizzato da Xamarin, permette la creazione di app iOS, Android e Mac, e include l'integrazione con gli SDK nativi, strumenti di debug e diverse funzionalità per la scrittura di codice.
- Visual Studio (Figura 2.15): offre un ambiente intuitivo e facilmente estendibile. Nel 2015, è stata rilasciata una versione gratuita, denominata Visual Studio Community, con l'occorrente per la creazione di app moderne mobile, desktop e web.

Come possiamo notare dalla Figura 2.14, su Mac OS X è possibile utilizzare Xamarin Studio solo per la creazione di app native e cross platform, fatta eccezione per Windows.

Recentemente (Novembre 2016), Microsoft ha rilasciato Visual Studio for Mac, offrendo, quindi, un IDE alternativo a Xamarin Studio per lo sviluppo di app Xamarin. Su Windows, invece, è possibile utilizzare Xamarin Studio solo per Android, mentre con Visual Studio non è possibile sviluppare app per Mac OS X, ed è necessario un Mac (raggiungibile in rete) per compilare e testare le app per iOS.

La scelta dell'IDE e del sistema operativo da utilizzare è, quindi, legata al modello di business previsto per la nostra app.

	MAC OS X	WINDOWS	
Development Environment	XAMARIN STUDIO	VISUAL STUDIO	XAMARIN STUDIO
Xamarin.iOS	Yes	Yes (with Mac computer)	No
Xamarin.Android	Yes	Yes	Yes
Xamarin.Forms	iOS & Android only	Android, Windows Phone, Windows (iOS with Mac computer)	Android only
Xamarin.Mac	Yes	No	No

Figura 2.14: Compatibilità di Visual Studio

A differenza dagli altri framework cross platform, fino a poco tempo fa l'utilizzo di Xamarin era vincolato alla sottoscrizione di uno specifico piano tariffario (Indie, Business, Enterprise). Con l'acquisizione da parte di Microsoft, Xamarin è diventato ufficialmente Open Source e gratuito. Viene offerto, quindi, pieno accesso al framework ed è permesso di utilizzarlo in tutte le versioni di Visual Studio (Figura 2.15).

Figura 2.15: Logo Visual Studio

3

Analisi dei requisiti e progettazione

In questo capitolo ci concentreremo, innanzitutto, sulla specifica e sull'analisi dei requisiti. Successivamente, illustreremo la progettazione dell'app evidenziando sia la progettazione delle applicazioni che quelle dell'interfaccia.

3.1 Analisi dei requisiti

3.1.1 Requisiti funzionali

L'applicazione deve poter consentire l'esecuzione delle seguenti attività:

- inserimento, durante il login, dei propri dati personali, quali nome, cognome, e-mail, cellulare, immagine del profilo ed eventuali intolleranze alimentari;
- modifica della lista delle intolleranze e dell'immagine del profilo in qualunque momento, anche dopo la creazione dell'account;
- organizzazione di un evento, che prevede, preliminarmente, l'inserimento di nome, luogo, data e orario dello stesso, più altre operazioni successive;
- creazione di una lista di invitati in grado di contenere sia persone che già sono autenticate, sia persone che non possiedono ancora un account e per cui è necessario l'inserimento manuale dei dati;
- visualizzazione delle eventuali intolleranze di tutti gli invitati;
- realizzazione di un menù che possa contenere sia piatti già proposti in eventi passati che altri del tutto nuovi;
- visualizzazione, al termine della realizzazione del menù, e prima di completare la procedura di creazione dell'evento, di tutte le occasioni, con specifica indicazione della data in cui alcune pietanze sono state proposte ai medesimi invitati e di cui forse l'organizzatore non ricorda;
- visualizzazione degli eventi a cui si è stati invitati e che sono stati rifiutati; tale visualizzazione verrà effettuata in una sezione "storia";
- visualizzazione di tutti gli eventi correnti, cioè quelli che ancora non hanno luogo; tale visualizzazione verrà effettuata in una sezione "attivi";
- visualizzazione di tutti i dati relativi ad un singolo evento;

- possibilità di accettare o rifiutare gli inviti ricevuti, anche dopo averli precedentemente accettati;
- possibilità per l'organizzatore di cancellare l'evento;
- possibilità di eliminare l'evento dalla lista degli eventi passati;
- inserimento, durante la registrazione, della regione attuale dell'utente;
- modifica della regione attuale;
- scelta tra "Dark theme" o "Light theme";
- scelta tra profilo pubblico o profilo privato;
- scelta dello stato di salute, tra "Malato" e "Sano";
- modifica di un evento creato precedentemente;
- scelta della disposizione degli invitati inserendo le misure del tavolo;
- modifica del nome dell'utente;
- modifica del numero di telefono;
- visualizzazione delle norme igieniche, delle regole per la ristorazione, delle informazioni aggiornate riguardanti la pandemia in Italia e delle normative per i colori delle regioni;
- invio e ricezione di una notifica per l'invito ad un evento appena creato;
- ricezione di una notifica, 5 ore prima dell'inizio prestabilito di un evento, per ricordare all'utente di indossare la mascherina e prendere tutte le precauzioni necessarie per impedire la diffusione del virus;
- ricezione di una notifica per avvertire che la regione attuale dell'utente ha cambiato colore.

3.1.2 Requisiti non funzionali

L'applicazione deve soddisfare le seguenti proprietà:

- obbligo di registrarsi prima di accedere all'applicazione;
- obbligo dell'indicazione dei campi nome, e-mail e cellulare per la creazione di un account (non vi è un controllo di sicurezza sul telefono e l'e-mail per verificare che quelli inseriti siano effettivamente esistenti o se si sta inserendo un'informazione "falsa");
- obbligo di inserimento di una e-mail valida (quindi contenente i caratteri "@" e ".");
- obbligo di inserimento di una password di almeno 6 caratteri;
- obbligo per l'utente di scegliere le proprie intolleranze da un nutrito gruppo prestabilito, senza possibilità di inserimento manuale;
- obbligo di inserimento dei campi relativi al nome, all'indirizzo, alla data e all'ora per la creazione di un evento;
- impossibilità per l'utente di organizzare eventi in date antecedenti a quella odierna;
- obbligo che la lista di invitati relativa ad un evento contenga almeno un invitato;
- obbligo per l'utente di creare un menù che contenga almeno una tipologia di portata;
- necessità dell'utilizzo di un database online per la memorizzazione dei dati relativi agli utenti e agli eventi; necessità di rendere possibile ad ogni utente l'accesso ai contatti che si desidera invitare e agli eventi di cui si vogliono visualizzare le informazioni;

- obbligo di compilare l'autocertificazione per partecipare ad un evento;
- obbligo di indicare la regione attuale (oltre ai campi nome, e-mail e cellulare già presenti nell'applicazione precedente) nella registrazione di un nuovo account;
- possibilità di posizionamento di più invitati vicini nel caso appartengano allo stesso nucleo familiare;
- impossibilità di posizionamento di più invitati vicini nel caso non appartengano allo stesso nucleo familiare;
- divieto di creare un evento se l'utente è malato;
- diveto di invitare un utente malato;
- divieto di creare un evento se l'utente si trova in zona rossa;
- divieto di invitare un utente che si trova in zona rossa;
- divieto di invitare un utente che si trova in una regione arancione diversa dalla propria;
- divieto di invitare un utente che si trova in regione arancione se l'organizzatore si trova in zona gialla;
- obbligo di inserimento della mail o del numero di cellulare completo se si vuole invitare un utente privato.

3.1.3 Casi d'uso

I casi d'uso sono, semplicemente, le funzionalità che un sistema fornisce, ovvero le operazioni che un utente, dall'esterno, può eseguire. Condizione necessaria affinché una funzionalità venga considerata come un caso d'uso è che abbia una relazione diretta con l'utente. Il punto di partenza di un progetto è costituito normalmente dagli use case diagram, o diagrammi dei casi d'uso, che illustrano in maniera semplice quello che il sistema può fare. Gli use case diagram rappresentano, quindi, una vista esterna del sistema.

Essi vengono realizzati mediante i seguenti costrutti:

- *Casi d'uso:* i casi d'uso rappresentano sequenze d'azioni svolte dal sistema.
- *Attori:* gli attori (ruoli) rappresentano persone (o altri sistemi) che interagiscono con il sistema.
- *Associazioni:* le associazioni tra attore e caso d'uso rappresentano il fatto che l'attore può "interagire" con il caso d'uso.

Nella Figura 3.1 vengono visualizzati i casi d'uso relativi all'applicazione da noi realizzata.

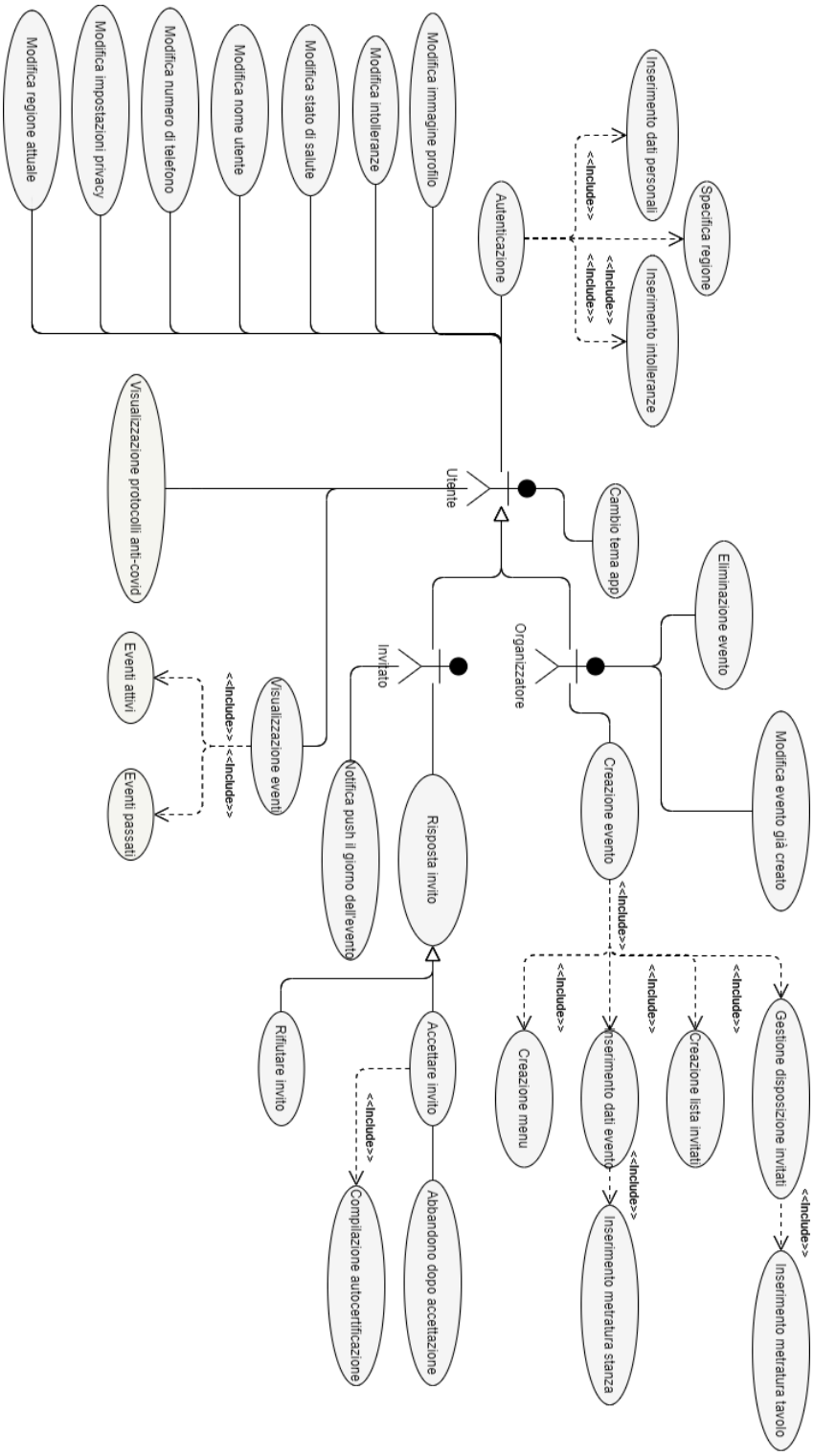


Figura 3.1: Diagramma dei casi d'uso

3.2 Progettazione dell'applicazione

3.2.1 Componente applicativa

Durante la fase di progettazione iniziale dell'applicazione abbiamo dovuto considerare le funzionalità che il nostro progetto doveva soddisfare partendo dallo studio delle caratteristiche che la sua struttura avrebbe dovuto presentare (mappa dell'app) fino ad arrivare alla progettazione del suo aspetto grafico (mockup).

Mappa dell'applicazione

Attraverso la mappa dell'applicazione è stata rappresentata schematicamente la struttura del nostro progetto al fine di consentire la visualizzazione dei suoi contenuti, distinti nelle diverse pagine dell'applicazione, con indicazione delle modalità di accesso per l'utente. Nella mappa mostrata in Figura 3.2 vengono riportate le funzionalità garantite dal nostro sistema.

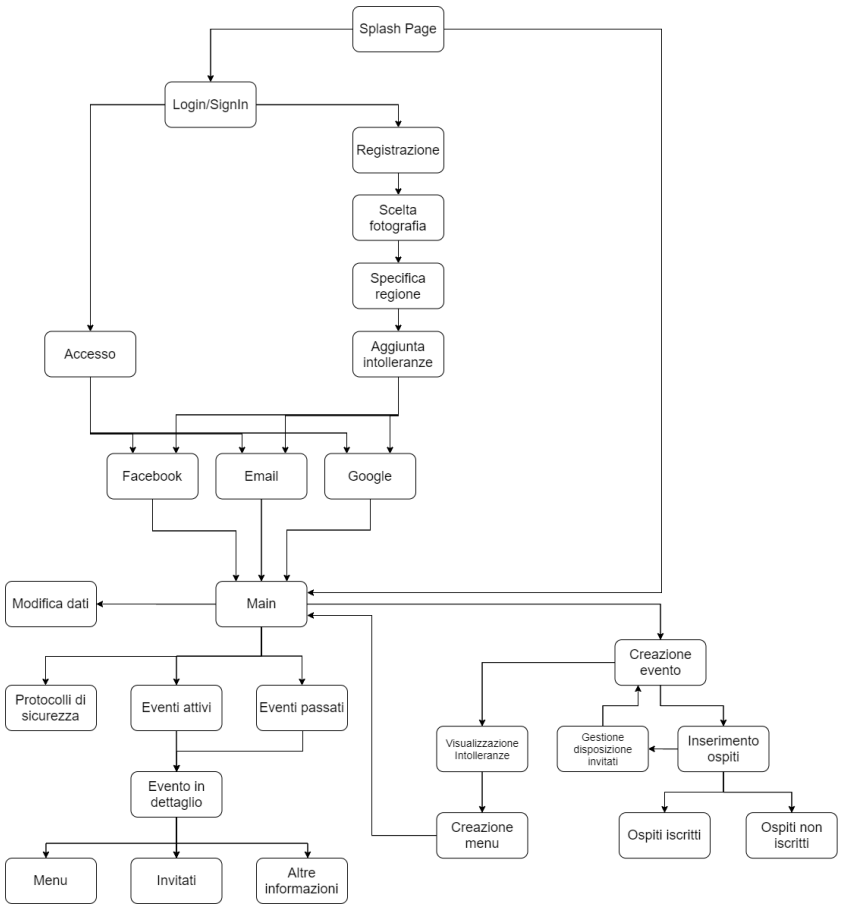


Figura 3.2: Mappa dell'applicazione

Diagrammi di attività

Per progettare le funzionalità fornite dalla nostra applicazione sono stati utilizzati i diagrammi di attività. Grazie a tali diagrammi possiamo organizzare più entità in un insieme di azioni secondo un determinato flusso, identificando le variazioni di stato al verificarsi di alcune condizioni legate ad una o più entità. Nelle Figure 3.3-3.7 vengono riportati i diagrammi di attività principali relativi alla nostra applicazione.

Il primo diagramma nella Figura 3.3 rappresenta il percorso di registrazione, in esso inizialmente si verifica se effettivamente l'utente non è mai stato registrato in precedenza, successivamente si passa all'inserimento di tutti i dati, quali: numero di cellulare (obbligatorio), foto del profilo (facoltativo), regione attuale (obbligatorio), intolleranze (facoltativo). Infine, si sceglie la modalità di accesso e, quindi, si completa il profilo inserendo nome, email e password, ovviamente tutti obbligatori.

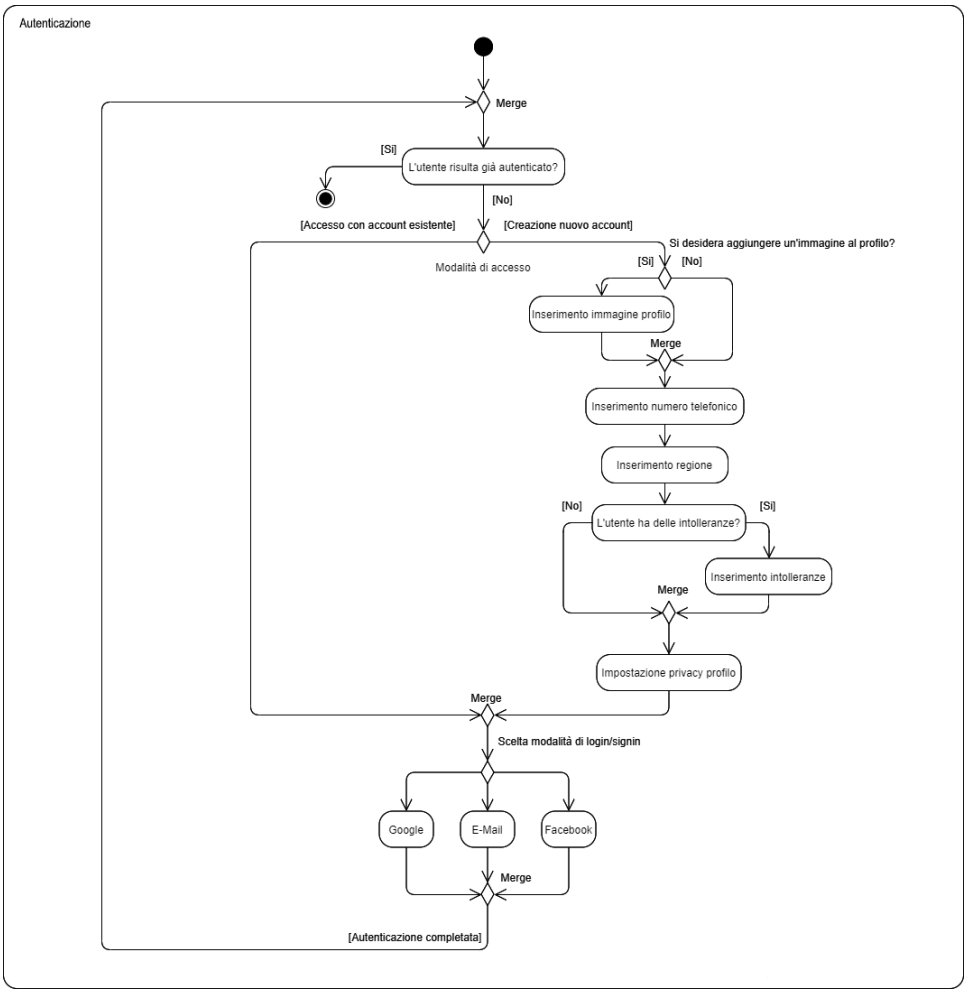
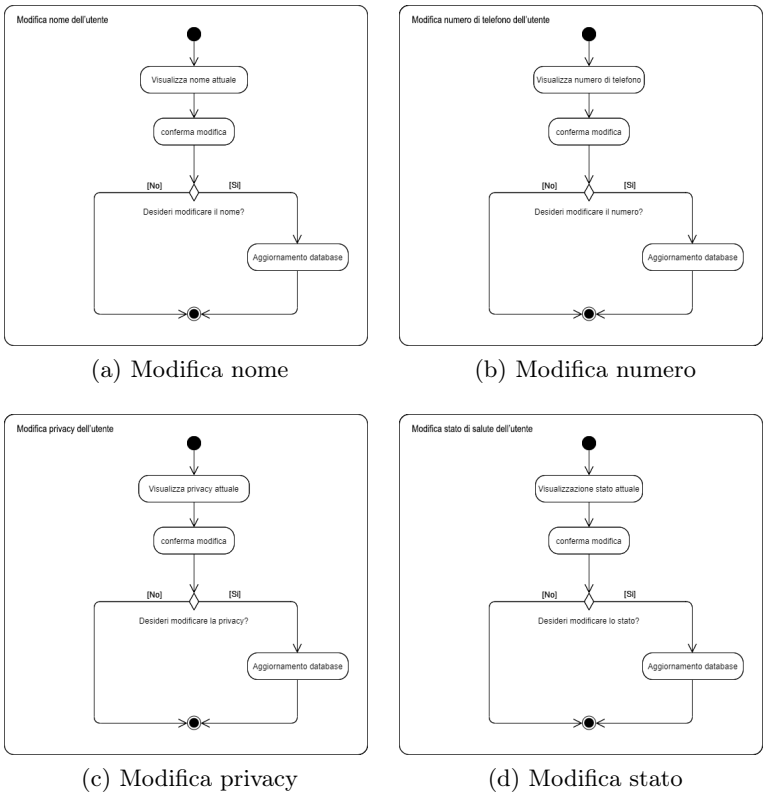


Figura 3.3: Diagramma di attività relativo all'autenticazione

Nelle Figure 3.4a e 3.5 vengono rappresentati i diagrammi di attività per la modifica dei dati del proprio profilo.



(a) Modifica nome (b) Modifica numero

(c) Modifica privacy (d) Modifica stato

Figura 3.4: Diagrammi di attività relativi alla modifica del nome, del numero di telefono, della privacy e dello stato

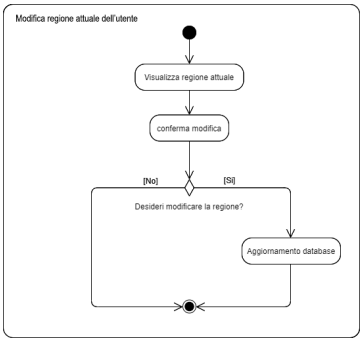


Figura 3.5: Diagramma di attività relativo alla modifica della regione

Nella Figura 3.6 vengono mostrati i passaggi per la risposta di un invito ad un evento. Quando si seleziona un evento possiamo scegliere se accettare o rifiutare l'invito; in caso di rifiuto, l'evento viene inserito direttamente negli "eventi passati". Nel caso, invece, l'utente accetti, viene richiesta la compilazione di un'autocertificazione per attestare che effettivamente egli sta bene e può partecipare all'evento, una volta compilata tale autocertificazione, e una volta accertato lo stato di salute, l'utente potrà finalmente partecipare all'evento.

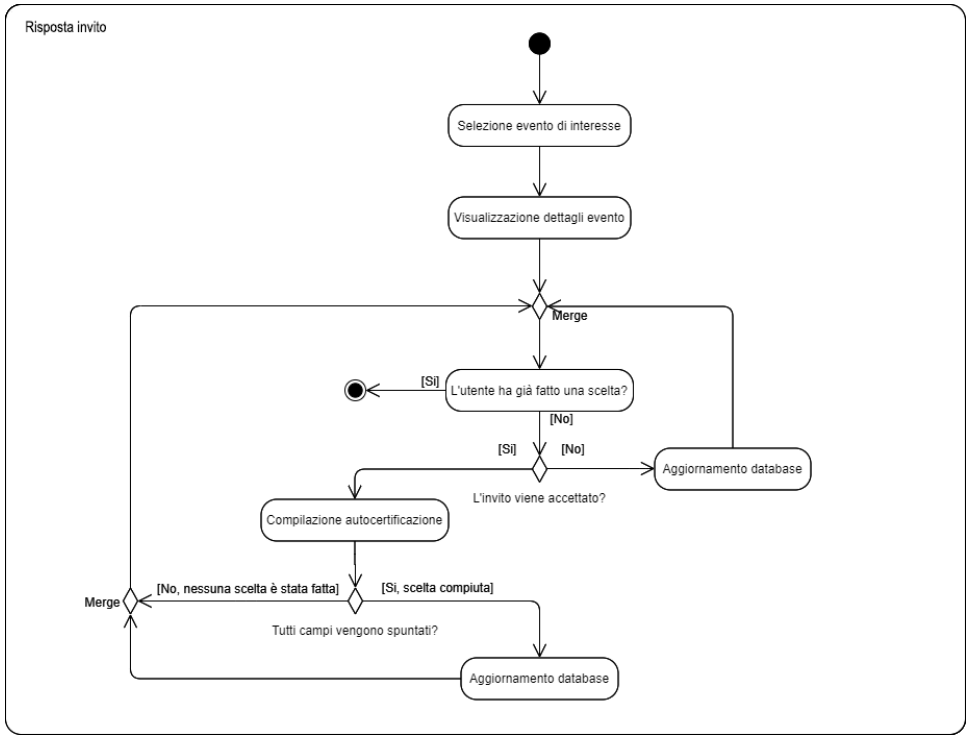


Figura 3.6: Diagramma di attività relativo alla risposta all'invito

Nello schema di Figura 3.7 vengono rappresentate le diverse fasi per l'invito di un utente. Prima di tutto si sceglie se invitare un utente già registrato all'interno dell'app oppure se si vuole creare un invitato bot per tener conto di quell'ospite. Nel caso in cui vogliamo creare un invitato bot, dovremo inserire tutte le informazioni (incluse le intolleranze) dell'ospite e creeremo un "falso" invitato. Se, invece, scegliamo di invitare una persona già registrata, possiamo cercarla normalmente scorrendo la lista oppure, se l'utente ha un account privato, dovremo cercarlo inserendo per intero la sua mail o il suo numero di telefono. Dopo aver scelto l'utente da invitare, quando questo verrà selezionato, potrebbero apparire alcuni alert che ci indicheranno che egli non può essere invitato. Ciò, ad esempio, accade nel caso in cui l'utente ha come stato di salute "malato", oppure se l'utente si trova in zona rossa o in una zona arancione mentre noi ci troviamo in zona gialla.

Una volta terminata la selezione degli ospiti, si passa alla disposizione intorno ai tavoli. Quando si inserisce un tavolo, è necessario specificare la sua lunghezza e la sua larghezza. Dopodichè si iniziano a trascinare gli ospiti facendo attenzione a non posizionarli in due quadretti vicini (cioè a meno di un metro l'uno dall'altro); qualora, si è in presenza di quest'ultimo tipo di posizionamento, si chiede all'organizzatore se i due utenti appartengano allo stesso nucleo familiare e, solo in quel caso, si potranno posizionare vicini.

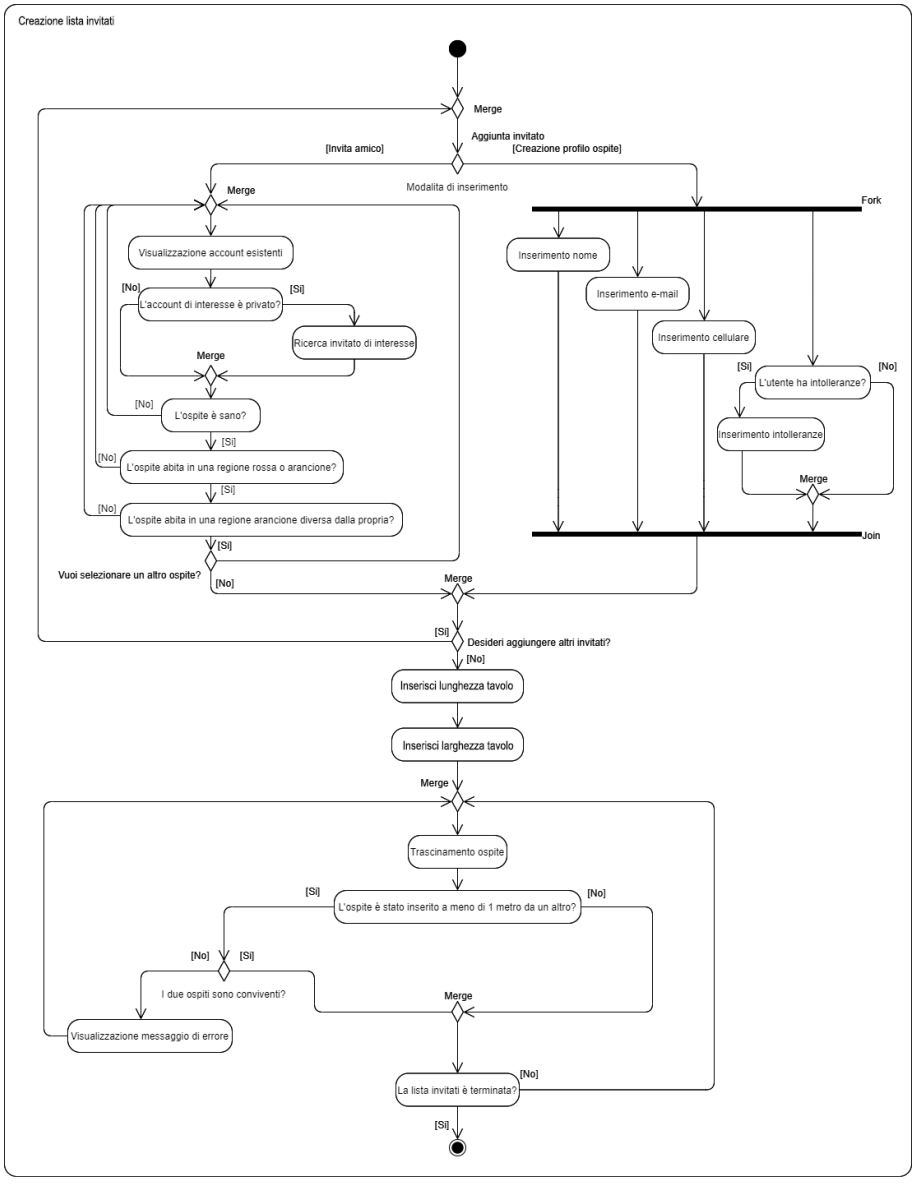


Figura 3.7: Diagramma di attività relativo alla creazione della lista degli invitati

Mockup

Per mezzo dei mockup è stato rappresentato, almeno in forma semplificata, l'aspetto che, al termine del lavoro, le singole pagine dell'applicazione dovranno presentare. Esistono diversi tipi di mockup, ciascuno con un livello di complessità e di dettaglio superiori al precedente. Nelle Figure 3.8-3.19 sono riportati i mockup di livello 0 realizzati con il software Balsamiq.

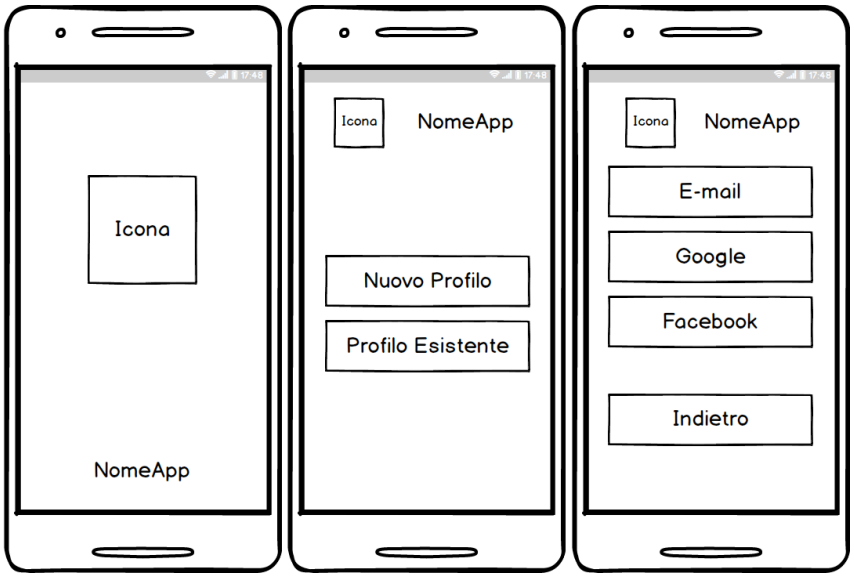


Figura 3.8: Mockup della SplashPage e del Login

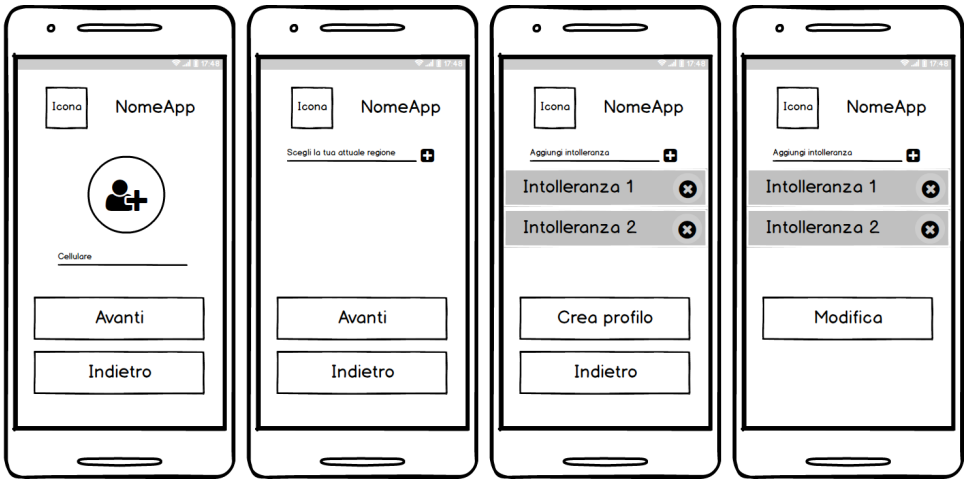


Figura 3.9: Mockup delle pagine per la registrazione

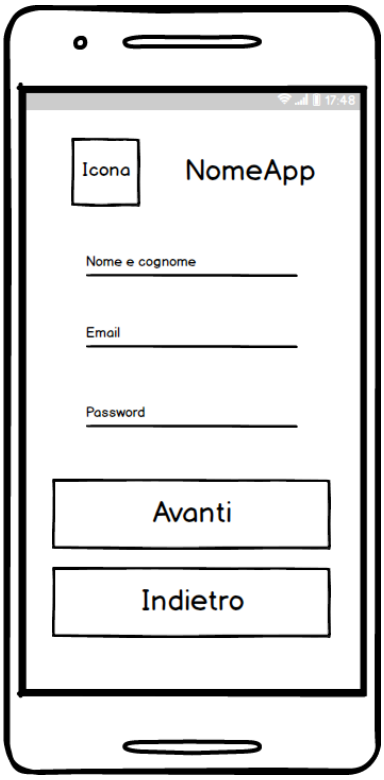


Figura 3.10: Mockup per la registrazione e l'accesso tramite email

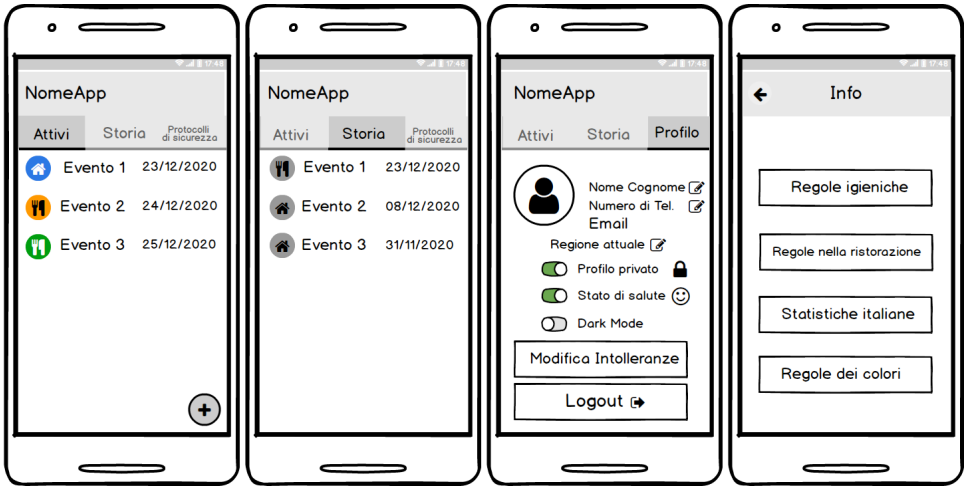


Figura 3.11: Mockup della MainPage e della InfoPage

Figura 3.12: Mockup dell'evento a cui l'utente è stato invitato prima della conferma

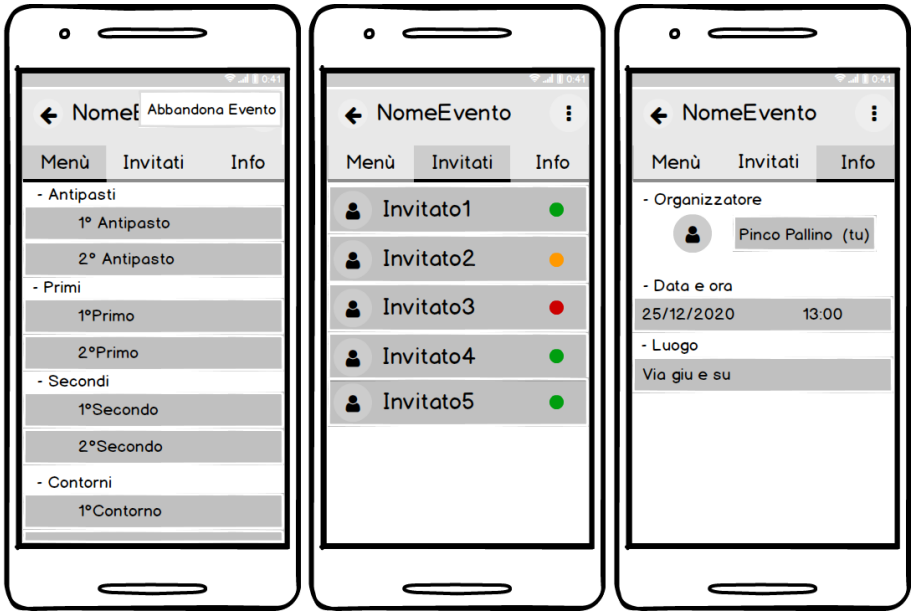


Figura 3.13: Mockup dell'evento a cui l'utente è stato invitato dopo la conferma

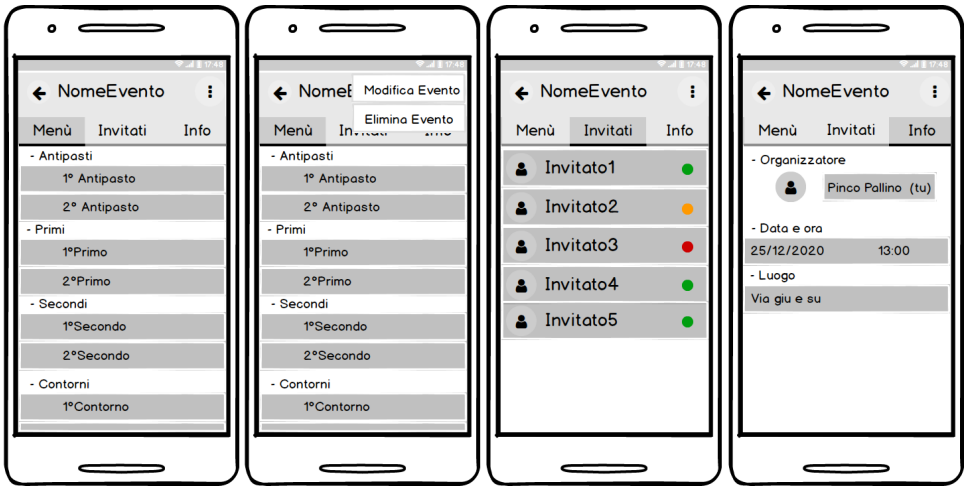


Figura 3.14: Mockup dell'evento organizzato dall'utente

Figura 3.15: Mockup della NewEventPage

Figura 3.16: Mockup delle pagine per l'aggiunta degli invitati

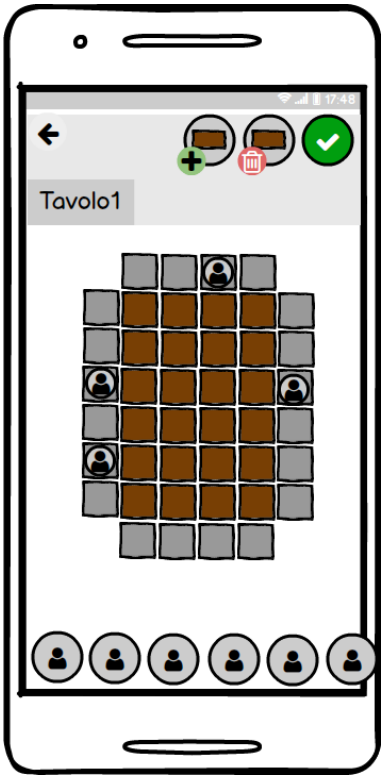


Figura 3.17: Mockup della pagina per il posizionamento degli invitati

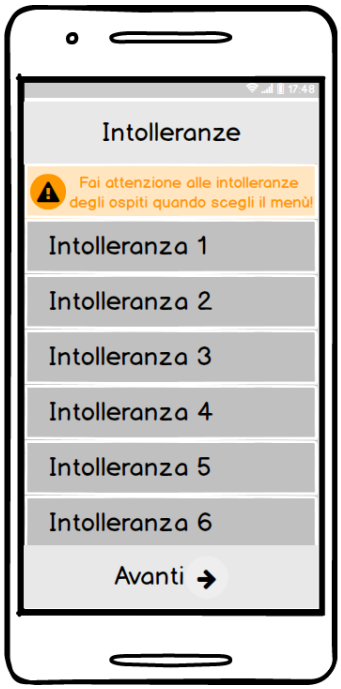


Figura 3.18: Mockup della pagina che racchiude tutte le intolleranze degli invitati

Figura 3.19: Mockup delle pagine per la creazione del menu

3.2.2 Componente dati

Questa applicazione necessita di un database online per poter funzionare; è per questa ragione che abbiamo usufruito dei servizi forniti dalla piattaforma di sviluppo Firebase.

Firebase

Tra i vari servizi abbiamo utilizzato i seguenti:

- *Authentication*, per semplificare l'implementazione della registrazione e del login, tramite e-mail e password, Google, oppure Facebook.
- *Storage*, per memorizzare le immagini di profili inserite dagli utenti in fase di registrazione, o in seguito, tutte identificate da un URL univoco.
- *Cloud Firestore*, un database NoSQL per memorizzare e sincronizzare dati sia dal lato client sia dal lato server.

In quanto NoSQL, Firestore è un database non relazionale, cioè non possiede una struttura rigida dove i dati vengono salvati all'interno di tabelle prestabilite. Al contrario, è costituito da collezioni; quest'ultime presentano al loro interno documenti in formato JSON, ciascuno con un Id univoco per distinguerlo dagli altri, ed una serie di campi, di numero non fisso ma variabile, che contengono i dati. A prima vista può sembrare che questa tipologia di database sia priva di qualunque regola o struttura. Queste ultime, in realtà, sono determinate dall'implementazione del client (nel nostro caso l'applicazione). I vantaggi rispetto ai database SQL si apprezzano soprattutto in scalabilità e prestazioni; i database SQL, infatti, per poter eseguire qualunque modifica, devono prima verificare che non vengano compromessi i vincoli (come, ad esempio, quelli di integrità dei dati) che ne determinano la correttezza, e quelli di integrità referenziale, che determinano i collegamenti tra le varie tabelle. Come già argomentato, il nostro database non presenta una struttura rigida; per questa ragione, non abbiamo ritenuto necessario seguire in maniera rigorosa i passaggi della progettazione concettuale, per arrivare a definire il diagramma ER, e della progettazione logica, per arrivare al modello relazionale finito. Il nostro database, inoltre, ha delle regole ben definite riguardanti l'accessibilità, e quindi la lettura e la scrittura (Figura 3.20).

Figura 3.20: Regole R/W del database

Nella Figura 3.21 viene riportato l'insieme delle collezioni del database con l'insieme dei campi che caratterizzano i documenti di ciascuna. Abbiamo la collezione `Event` che raccoglie tutti gli eventi organizzati tramite la nostra app. Ogni evento ha come campi:

- `nomeOrganizzatore`: contiene il nome dell'organizzatore;
- `emailOrganizzatore`: contiene l'email dell'organizzatore;
- `phoneOrganizzatore`: contiene il numero di telefono dell'organizzatore;
- `idOrganizzatore`: contiene l'id, univoco, dell'utente organizzatore;
- `eventName`: contiene il nome dell'evento;
- `idEvent`: contiene l'id, univoco, per ogni evento;
- `address`: contiene l'indirizzo dell'evento;
- `data`: contiene la data in cui si farà l'evento;
- `ora`: contiene a che ora si farà l'evento;
- `invitedGuests`: contiene le map degli ospiti bot che vengono descritti dai campi:
  - `nameGuest`: contiene il nome dell'ospite bot;
  - `emailGuest`: contiene la mail dell'ospite bot;
  - `phoneGuest`: contiene il numero di telefono dell'ospite bot;
  - `intolerancesGuest`: contiene le intolleranze dell'ospite bot.
- `invitedContacts`: contiene le map di tutti gli invitati; questi ultimi vengono descritti dai seguenti campi:
  - `accettato`: riporta la risposta dell'invitato (true se ha accettato, false se ha rifiutato);
  - `IdGuest`: contiene l'id, univoco, di ogni utente invitato;
  - `scelta`: contiene un booleano che dichiara se l'utente ha fatto o no la sua scelta (true se ha scelto, false se non ha ancora deciso).
- `meals`: contiene il menù dell'evento; quest'ultimo viene descritto dai seguenti campi:
  - `antipasti`: contiene la lista di tutti gli antipasti scelti dall'organizzatore;
  - `primi`: contiene la lista di tutti i primi scelti dall'organizzatore;
  - `secondi`: contiene la lista di tutti i sceondi scelti dall'organizzatore;
  - `contorni`: contiene la lista di tutti i contorni scelti dall'organizzatore;
  - `dolci`: contiene la lista di tutti i dolci scelti dall'organizzatore;
  - `bevande`: contiene la lista di tutte le bevande scelte dall'organizzatore.

Nella collezione `Utilities` vi è la lista di tutte le intolleranze tra le quali l'utente può selezionare quelle che lo riguarda.

Infine, nella collezione `Person`, vi sono tutti i documenti collegati agli utenti iscritti alla nostra applicazione; tale collezione prevede i seguenti campi:

- `name`: contiene il nome dell'utente;
- `email`: contiene l'email dell'utente;
- `phone`: contiene il numero di telefono dell'utente;
- `uriProfile`: contiene il nome con cui è stata salvata l'immagine di profilo dell'utente all'interno dello storage di firebase;
- `region`: contiene la regione attuale dell'utente;
- `malato`: contiene lo stato dell'utente (true se è malato, false se è sano);
- `privato`: contiene la privacy dell'account dell'utente (true se è privato, false se è pubblico);

- `token`: contiene il token dell'utente per l'invio e la ricezione delle notifiche;
- `intolleranze`: contiene tutte le intolleranze dell'utente;
- `idEventi`: contiene tutti gli id, univoci, degli eventi che l'utente ha organizzato o a cui è stato invitato, che ha accettato o a cui non ha ancora risposto;
- `idEventiPassati`: contiene tutti gli id, univoci, degli eventi passati o che l'utente ha rifiutato;
- `dishes`: contiene tutti i piatti che l'utente ha preparato durante i suoi eventi e che vengono divisi in antipasti, primi, secondi, contorni, dolci e bevande;

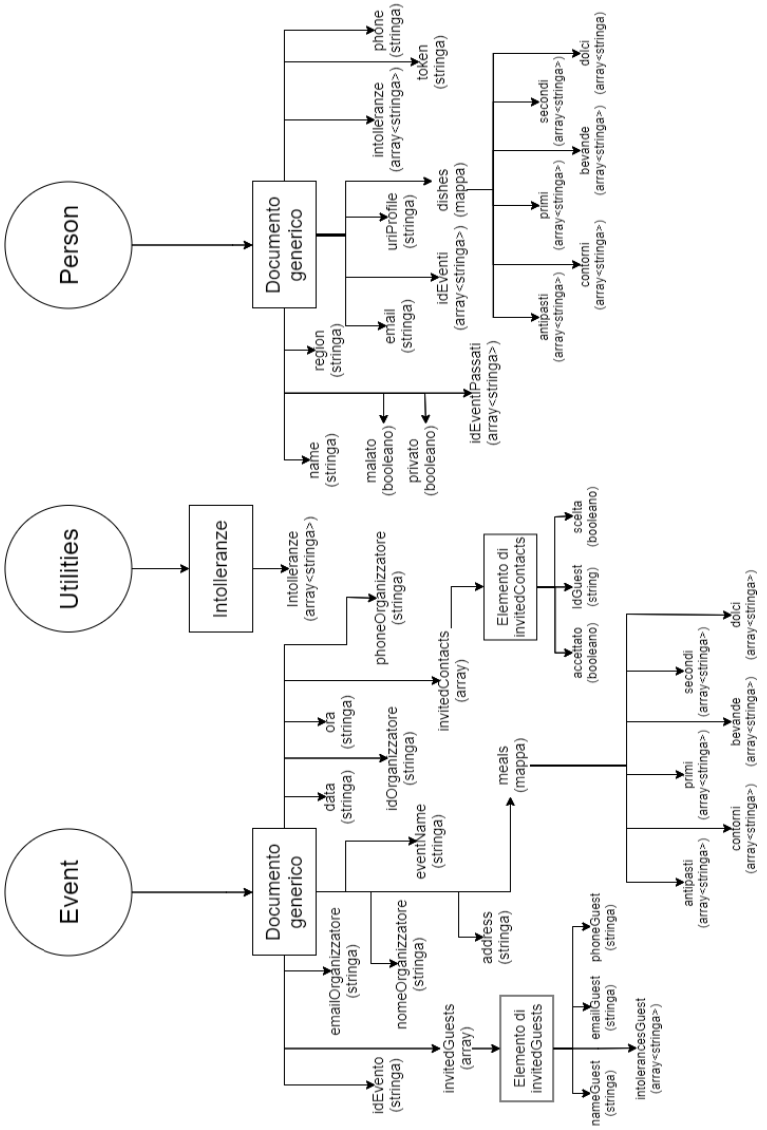


Figura 3.21: Schema della struttura del database Firestore

4

Implementazione dell'applicazione

In questo capitolo vedremo come è stata strutturata l'applicazione, le singole pagine che la compongono e tutto ciò che si trova all'interno della soluzione.

4.1 Struttura del progetto - Soluzione

Il progetto, sviluppato tramite l'IDE Visual Studio 2019, è strutturato come descritto nella Figura 4.1. Il linguaggio XAML è stato utilizzato per la definizione dei layout di ogni pagina ed il linguaggio di programmazione C# per definire il comportamento.

Visto che la nostra app ibrida è stata concepita per funzionare solo su dispositivi Android, la soluzione è costituita da due soli progetti:

- `HUP.Forms`;
- `HUP.Android`.

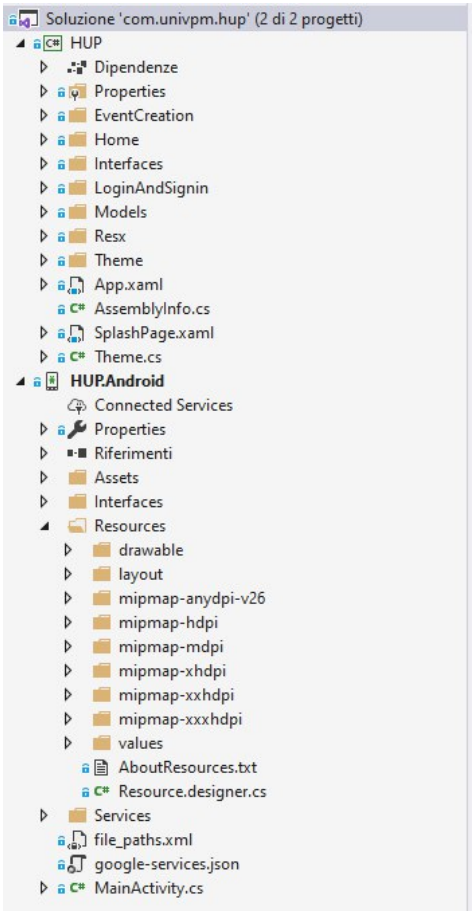


Figura 4.1: Struttura della soluzione dell'applicazione

4.2 HUP.Forms

La sezione *HUP.Forms* è il progetto cross-platform contenente la logica di business, l'accesso ai dati e l'interfaccia utente realizzata in Xamarin.Forms. Questa sezione è divisa nelle cartelle:

- `EventCreation`;
- `Popup`;
- `Home`;
- `Interfaces`;
- `LoginAndSignin`;
- `Models`;
- `Resx`;
- `Theme`;

4.2.1 App

Nel file `App.xaml.cs` vi sono le informazioni principali per far partire l'applicazione, come la scelta del tema iniziale, la pagina con cui deve avviarsi l'applicazione e l'abilitazione del flag per la funzione (che è, ancora in fase sperimentale da parte degli sviluppatori Xamarin) del DragAndDrop. Nel file `app.xaml`, invece, abbiamo il `ResourceDictionary`, dove vengono specificati i colori iniziali di tutte le componenti dell'app impostate come `DynamicResource` in modo che possano cambiare in base al tema scelto.

4.2.2 EventCreation

Nella cartella `EventCreation` sono contenute tutte le pagine riguardanti la creazione e la modifica degli eventi. Tali pagine vengono descritte nel seguito di questa sezione.

NewEvent

Quando si vuole creare o modificare un evento questa è la prima pagina che viene mostrata per l'inserimento (o la modifica) di tutte le informazioni primarie dell'evento stesso, come data, ora, nome e luogo. Nel Listato 4.1 abbiamo il codice che gestisce la memorizzazione delle informazioni riguardanti l'evento che inseriamo, mentre nel file `NewEvent.xaml` vi è la progettazione di tutta la parte grafica della pagina.

```
private void GetData(ObservableCollection<Guest> guestList){
    var list = guestList.OrderBy(i => i.Name);
    GuestList = new ObservableCollection<Guest>(list);
    GuestListView.ItemsSource = GuestList;}
//metodo lanciato al click del bottone invita amico
private async void InviteFriendButton_Clicked(object sender, EventArgs e){
    if (!ModifyEvent)
    {await Navigation.PushAsync(new ExistingContact(GuestList));}
    else {await Navigation.PushAsync(new ExistingContact(GuestList, ModifyEvent, EventoDaModificare));}}
//metodo lanciato al click del bottone aggiungi prfilo ospite
private async void CreateGuestProfile_Clicked(object sender, EventArgs e){
    if (!ModifyEvent)
    {await Navigation.PushAsync(new NewContact(GuestList));}
    else {await Navigation.PushAsync(new NewContact(GuestList, ModifyEvent, EventoDaModificare));}}
//metodo lanciato per tornare nella pagina NewEvent con la lista
private async void DoneButton_Clicked(object sender, EventArgs e){
    if (!ModifyEvent)
    {await Navigation.PushAsync(new NewEvent(GuestList));}
    else {await Navigation.PushAsync(new NewEvent(GuestList, ModifyEvent, EventoDaModificare));}}
//metodo lanciato al click del bottone rimuovi all'interno di ogni elemento della Listview
private async void DeleteButton_Clicked(object sender, EventArgs e){
    bool response = await DisplayAlert(AppResources.RemoveGuestAlertTitle, AppResources.
           RemoveGuestAlertBody, AppResources.yes, AppResources.no);
    if (response){
        var button = sender as ImageButton;
        var guest = button.BindingContext as Guest;
        GuestList.Remove(guest);
        if (ModifyEvent){
            if (!(string.IsNullOrEmpty(guest.Id))){
                var myDocument = await CrossCloudFirestore.Current.Instance.GetCollection("Person").
                       GetDocument(guest.Id).GetDocumentAsync();
                var myData = myDocument.Data;
                IList EventiInvitato = (IList)myData["idEventi"];
                for (int i = 0; i < EventiInvitato.Count; i++){
                    if (EventiInvitato[i].Equals(EventoDaModificare.IdEvento)){
                        EventiInvitato.Remove(EventiInvitato[i]);
                        await CrossCloudFirestore.Current.Instance.GetCollection("Person").GetDocument(guest.
                               Id).UpdateDataAsync("idEventi", EventiInvitato);}}}}
        GuestListSearchBar.Text = "";
        DependencyService.Get<IMessage>().ShortAlert(guest.Name + AppResources.RemovedGuestToast);}}
//metodo lanciato ogni vlta che il testo della searchbar viene modificato
private void GuestListSearchBar_TextChanged(object sender, TextChangedEventArgs e){
    var searchText = e.NewTextValue;
    if (string.IsNullOrWhiteSpace(searchText)){
        GuestListView.ItemsSource = GuestList; }else{
        GuestListView.ItemsSource = GuestList.ToList().Where(p => p.Name.Contains(searchText) || p.Email.
               Contains(searchText) || p.Phone.Contains(searchText));}}
```

```
//metodo lanciato al click del tasto cerca della SearchBar
private void GuestListSearchBar_SearchButtonPressed(object sender, EventArgs e){
    var searchText = GuestListSearchBar.Text;
    if (string.IsNullOrWhiteSpace(searchText))
    {GuestListView.ItemsSource = GuestList;}
    else{ GuestListView.ItemsSource = GuestList.ToList().Where(p => p.Name.Contains(searchText) || p.Email.
           Contains(searchText) || p.Phone.Contains(searchText));}}
```

Listato 4.1: Code-Behind della pagina `NewEvent.xaml.cs`

ContactEntry

Questa pagina permette di gestire la lista degli invitati. Nel Listato 4.2 mostriamo il codice che gestisce la lista degli invitati, mentre nel file `ContactEntry.xaml` vi è la progettazione di tutta la parte grafica della pagina.

```
private void GetData(ObservableCollection<Guest> guestList){
    var list = guestList.OrderBy(i => i.Name);
    GuestList = new ObservableCollection<Guest>(list);
    GuestListView.ItemsSource = GuestList;}
//metodo lanciato al click del bottone invita amico
private async void InviteFriendButton_Clicked(object sender, EventArgs e){
    if (!ModifyEvent)
    {await Navigation.PushAsync(new ExistingContact(GuestList));}
    else {await Navigation.PushAsync(new ExistingContact(GuestList, ModifyEvent, EventoDaModificare));}}
//metodo lanciato al click del bottone aggiungi prfilo ospite
private async void CreateGuestProfile_Clicked(object sender, EventArgs e){
    if (!ModifyEvent)
    {await Navigation.PushAsync(new NewContact(GuestList));}
    else
    {await Navigation.PushAsync(new NewContact(GuestList, ModifyEvent, EventoDaModificare));}}
//metodo lanciato per tornare nella pagina NewEvent con la lista
private async void DoneButton_Clicked(object sender, EventArgs e){
    if (!ModifyEvent)
    {await Navigation.PushAsync(new NewEvent(GuestList));}
    else
    {await Navigation.PushAsync(new NewEvent(GuestList, ModifyEvent, EventoDaModificare));}}
//metodo lanciato al click del bottone rimuovi all'interno di ogni elemento della Listview
private async void DeleteButton_Clicked(object sender, EventArgs e){
    bool response = await DisplayAlert(AppResources.RemoveGuestAlertTitle, AppResources.
           RemoveGuestAlertBody, AppResources.yes, AppResources.no);
    if (response){
        var button = sender as ImageButton;
        var guest = button.BindingContext as Guest;
        GuestList.Remove(guest);
        if (ModifyEvent){
            if (!(string.IsNullOrEmpty(guest.Id))){
                var myDocument = await CrossCloudFirestore.Current.Instance.GetCollection("Person").
                       GetDocument(guest.Id).GetDocumentAsync();
                var myData = myDocument.Data;
                IList EventiInvitato = (IList)myData["idEventi"];
                for (int i = 0; i < EventiInvitato.Count; i++){
                    if (EventiInvitato[i].Equals(EventoDaModificare.IdEvento)){
                        EventiInvitato.Remove(EventiInvitato[i]);
                        await CrossCloudFirestore.Current.Instance.GetCollection("Person").GetDocument(guest.
                               Id).UpdateDataAsync("idEventi", EventiInvitato);}}}}
        GuestListSearchBar.Text = "";
        DependencyService.Get<IMessage>().ShortAlert(guest.Name + AppResources.RemovedGuestToast);}}
//metodo lanciato ogni vlta che il testo della searchbar viene modificato
private void GuestListSearchBar_TextChanged(object sender, TextChangedEventArgs e){
    var searchText = e.NewTextValue;
    if (string.IsNullOrWhiteSpace(searchText))
    {GuestListView.ItemsSource = GuestList;}
    else
    {GuestListView.ItemsSource = GuestList.ToList().Where(p => p.Name.Contains(searchText) || p.Email.
           Contains(searchText) || p.Phone.Contains(searchText));}}
//metodo lanciato al click del tasto cerca della SearchBar
private void GuestListSearchBar_SearchButtonPressed(object sender, EventArgs e){
    var searchText = GuestListSearchBar.Text;
    if (string.IsNullOrWhiteSpace(searchText)){
        GuestListView.ItemsSource = GuestList;
        }else{
        GuestListView.ItemsSource = GuestList.ToList().Where(p => p.Name.Contains(searchText) || p.Email.
               Contains(searchText) || p.Phone.Contains(searchText));}}
```

Listato 4.2: Code-Behind della pagina `ContactEntry.xaml.cs`

ExistingContact

In questa pagina è possibile visualizzare la lista degli utenti esistenti e aggiungerli a quella degli ospiti. Nel Listato 4.3 riportiamo le parti fondamentali del codice che gestisce tutta la logica della pagina, mentre nel file `ExistingContact.xaml` vi è la progettazione di tutta la parte grafica della stessa.

```
//metodo per recuperare tutti gli utenti dal db, fa utilizzo di una libreria crossplatform per l'utilizzo
       del cloud firestore
private async void GetData(){
    //conversione di ciascun documento in oggetto Person e inserimento nella lista
    foreach (var document in group.Documents){
        var dataDictionary = document.Data;}
        //controllo per non inserire nella lista completa l'utente organizzatore dell'evento
        if (person.Id != myId){
            if ((person.Privato)&&(person.Id != myId))
            {ExistingPrivateContactList.Add(person);}
            else if((!(person.Privato))&&(person.Id != myId))
            {ExistingContactList.Add(person);}}}
    await ExistingContactProgressBar.ProgressTo(.9, 250, Easing.Linear);
    //utilizzo della lista realizzata come ItemsSource della ListView e successiva rimozione della
         ProgressBar
    ExistingContactListView.ItemsSource = ExistingContactList;
    await ExistingContactProgressBar.ProgressTo(1, 250, Easing.Linear);
    ExistingContactProgressBar.IsVisible = false;}
//implementazione funzionalita' di ricerca
private void ExistingContactSearchBar_TextChanged(object sender, TextChangedEventArgs e){
    var searchText = e.NewTextValue;
    if (string.IsNullOrWhiteSpace(searchText))
    {ExistingContactListView.ItemsSource = ExistingContactList;}
    else{
        //Nel caso stiamo cercando una persona se inserisci il nome o il nuemro o la mail parzialmente ti
               trovera' solo gli utenti con profilo pubblico
        //mentre se vuoi cercare un profilo privato devi necessariamente conoscere ed inserire per intero
               nella search bar o il numero di telefono o l'email
        ExistingContactListView.ItemsSource = ExistingSearchedContactList.Concat(ExistingContactList.ToList
               ().Where(p => p.Name.Contains(searchText) || p.Email.Contains(searchText) || p.Phone.
               Contains(searchText))).Concat(ExistingPrivateContactList.ToList().Where(p => p.Email.Equals(
               searchText) || p.Phone.Equals(searchText))).ToList();}}
private void ExistingContactSearchBar_SearchButtonPressed(object sender, EventArgs e){
    var searchText = ExistingContactSearchBar.Text;
    if (string.IsNullOrWhiteSpace(searchText))
    {ExistingContactListView.ItemsSource = ExistingContactList;}
    else{
         ExistingContactListView.ItemsSource = ExistingSearchedContactList.Concat(ExistingContactList.ToList
                ().Where(p => p.Name.Contains(searchText) || p.Email.Contains(searchText) || p.Phone.
                Contains(searchText))).Concat(ExistingPrivateContactList.ToList().Where(p => p.Email.Equals
                (searchText) || p.Phone.Equals(searchText))).ToList();}}
//metodo lanciato quando viene cliccata una delle CheckBox di una della righe della ListView
private async void CheckGuest_CheckedChanged(object sender, CheckedChangedEventArgs e){
          //Qui se stai in regione arancione e inviti qualcuno in zona gialla
          if (Preferences.Get(person.Regione, "nero").ToString().Equals("GIALLA")&& Preferences.Get(
                 dataDictionary2["region"].ToString(), "nero").ToString().Equals("ARANCIONE")){
              person.IsSelected = false;
              button.IsChecked = false;
              await DisplayAlert(AppResources.OrangeYellowAlertTitle, AppResources.OrangeYellowAlertBody,
                     AppResources.Ok);}
          //Qui se stai in regione gialla e inviti qualcuno in zona arancione
          else if (Preferences.Get(person.Regione, "nero").ToString().Equals("ARANCIONE") && Preferences.
                 Get(dataDictionary2["region"].ToString(), "nero").ToString().Equals("GIALLA")){
              person.IsSelected = false;
              button.IsChecked = false;
              await DisplayAlert(AppResources.YellowOrangeAlertTitle, AppResources.YellowOrangeAlertBody,
                     AppResources.Ok);}
          //Qui se stai in zona gialla o arancione ma inviti qualcuno in zona rossa
          else if (Preferences.Get(person.Regione, "nero").ToString().Equals("ROSSA")){
              person.IsSelected = false;
              button.IsChecked = false;
              await DisplayAlert(AppResources.OrangeOrYellowRedAlertTitle, AppResources.
                     OrangeOrYellowRedAlertBody, AppResources.Ok);}
          //Qui se inviti qualcuno in zona arancione diversa dalla tua
          else if (Preferences.Get(person.Regione, "nero").ToString().Equals("ARANCIONE") && Preferences.
                 Get(dataDictionary2["region"].ToString(), "nero").ToString().Equals("ARANCIONE") && !(
                 dataDictionary2["region"].ToString().Equals(person.Regione))){
              person.IsSelected = false;
              button.IsChecked = false;
              await DisplayAlert(AppResources.OrangeDifferentAlertTitle, AppResources.
                     OrangeDifferentAlertBody, AppResources.Ok);
          }else {SelectedContacts.Add(person);}}}
    else if (!person.IsSelected && SelectedContacts.Contains(person))
    {SelectedContacts.Remove(person);}
```

Listato 4.3: Code-Behind della pagina `ExistingContact.xaml.cs`

NewContact

Questa pagina viene utilizzata nel caso si voglia aggiungere un ospite che non abbia un account nell'applicazione, creando un invitato bot che servirà come promemoria per il conteggio degli invitati o per le sue intolleranze. Infatti, per creare questo contatto, si vanno ad inserire, oltre a nome e cognome, anche le sue intolleranze. Nel Listato 4.4 mostriamo il codice che gestisce tutta la parte di logica della pagina, mentre nel file `NewContact.xaml` vi è la progettazione di tutta la parte grafica della stessa.

```
//metodo per impostare i dati dell'AutoComplete prendendoli dal db
private async void SetupPicker(){
    var doc = await CrossCloudFirestore.Current.Instance.GetCollection("Utilities").GetDocument("
            Intolleranze").GetDocumentAsync();
    var list = (IList)doc.Data["Intolleranze"];
    var orderedList = new List<string>();
    foreach (string item in list)
    {orderedList.Add(item);}
    orderedList.Sort();
    IntolerancePicker.ItemsSource = orderedList;}
//metodi lanciati al variare dei valori nelle Entry per aggiornare la variabili corrispondenti
private void NameEntry_TextChanged(object sender, TextChangedEventArgs e){
    GuestName = e.NewTextValue;}
private void EmailEntry_TextChanged(object sender, TextChangedEventArgs e){
    GuestEmail = e.NewTextValue;}
private void PhoneEntry_TextChanged(object sender, TextChangedEventArgs e){
    GuestPhone = e.NewTextValue;}
//metodo lanciato al click per eleminare l'intolleranza selezionata
private async void DeleteButton_Clicked(object sender, EventArgs e){
    bool response = await DisplayAlert(AppResources.DeleteIntolerancesTitle, AppResources.
            DeleteIntolerancesBody, AppResources.yes, AppResources.no);
    if (response){
        var button = sender as ImageButton;
        var intolerance = button.BindingContext as Intolerance;
        GuestIntolerances.Remove(intolerance);}}
//metodo lanciato quando si seleziona uno degli elementi del picker in cui l'elemento selezionato viene
        aggiunto all'ItemsSource della ListView
private void IntolerancePicker_SelectedIndexChanged(object sender, EventArgs e){
    var picker = sender as Picker;
    int selectedIndex = picker.SelectedIndex;
    if (selectedIndex != -1){
        var intolerance = new Intolerance((string)IntolerancePicker.ItemsSource[selectedIndex]);
        if (!GuestIntolerances.Contains(intolerance)){
            GuestIntolerances.Add(intolerance);
            picker.SelectedIndex = -1;}}}
//metodo lanciato al click del bottone aggiungi ospiti per creare un Guest manualmente da aggiungere alla
        lista
private async void AddGuest_Clicked(object sender, EventArgs e){
    if ((GuestName != "") && (GuestEmail != "") && (GuestPhone != "")){
        var intolerances = new List<string>();
        foreach (var item in GuestIntolerances){
            intolerances.Add(item.Name);}
        Guest guest = new Guest(){
            Name = GuestName,
            Email = GuestEmail,
            Phone = GuestPhone,
            Intolleranze = intolerances,
            Accettato = false,
            Scelta = false,
            Id = "",
            UriProfile = "ic_person_bot_add_18dp.xml"};
        if (!CurrentList.Contains(guest))
        {CurrentList.Add(guest);}
        if (!ModifyEvent)
        {await Navigation.PushAsync(new ContactEntry(CurrentList));}
        else{await Navigation.PushAsync(new ContactEntry(CurrentList, ModifyEvent, EventoDaModificare));}}
    else if (GuestName == "")
    {DependencyService.Get<IMessage>().ShortAlert(AppResources.GuestNameMiss);}
    else if (GuestEmail == "")
    {DependencyService.Get<IMessage>().ShortAlert(AppResources.GuestEmailMiss);}
    else if (GuestPhone == "")
    {DependencyService.Get<IMessage>().ShortAlert(AppResources.GuestPhoneMiss);}}
```

Listato 4.4: Code-Behind della pagina `NewContact.xaml.cs`

FoodIntolerances

Questa pagina serve come riassunto di tutte le intolleranze degli invitati.

TabTable

Questa pagina serve come Tab base per la TabbedPage in cui vi è il posizionamento degli invitati. Infatti, ogni volta che si crea una nuova sezione in `GuestPositionTable`, si richiama questa pagina definendo, sulla base alle misure scelte dall'utente, le dimensioni del tavolo. Nel file `TabTable.xaml.cs` mostriamo il codice che gestisce tutta la parte di logica della pagina, mentre nel file `TabTable.xaml` vi è la progettazione di tutta la parte grafica della stessa.

GuestPositionTable

In questa pagina l'organizzatore dell'evento potrà posizionare gli ospiti intorno ai tavoli rispettando le normative Covid-19. Nel Listato 4.5 abbiamo il codice che gestisce tutta la parte di logica della pagina, mentre nel file `GuestPositionTable.xaml` vi è la progettazione di tutta la parte grafica della stessa.

```
private async void DoneButton(object sender, EventArgs e){
    bool resp = await DisplayAlert(AppResources.Confirm, AppResources.GuestDisposition, AppResources.yes,
          AppResources.Cancel);
    if (resp)
    {await Navigation.PushAsync(new FoodIntolerances(Nome, Luogo, Data, Ora, GuestList));}}
private async void TableButtonAddPressed(object sender, EventArgs e){
    saveLunghezza = await DisplayPromptAsync(AppResources.EnterLengthTitle, AppResources.EnterLengthBody,
          AppResources.Done, AppResources.Cancel, keyboard: Keyboard.Numeric);
    if (saveLunghezza != null){
        if (IsDigitsOnly(saveLunghezza)){
            saveLarghezza = await DisplayPromptAsync(AppResources.EnterWidthTitle, AppResources.
                  EnterWidthBody, AppResources.Done, AppResources.Cancel, keyboard: Keyboard.Numeric);
            if (saveLarghezza != null){
                if (IsDigitsOnly(saveLarghezza)){
                    if (saveLunghezza.Contains(",")){
                        string[] lunghezzaDivisa = saveLunghezza.Split(',');
                        saveLunghezza = lunghezzaDivisa[0] + "." + lunghezzaDivisa[1];}
                    if (saveLarghezza.Contains(",")){
                        string[] larghezzaDivisa = saveLarghezza.Split(',');
                        saveLarghezza = larghezzaDivisa[0] + "." + larghezzaDivisa[1];}
                    LarghezzaTavolo = Convert.ToInt32(saveLarghezza);
                    LunghezzaTavolo = Convert.ToInt32(saveLunghezza);
                    if (count == 0){
                        count = 1;
                        listaNumeroTavoli.Add(count);
                    }else{
                        for (int i = 1; i <= 1000; i++){
                            if (!(listaNumeroTavoli.Contains(i))){
                                count = i;
                                listaNumeroTavoli.Add(count);
                                break;}}}
                    this.Children.Add(new TabTavolo(LunghezzaTavolo, LarghezzaTavolo, GuestList) { Title =
                          AppResources.table + count.ToString() });}}}}}
bool IsDigitsOnly(string str){
    foreach (char c in str){
        if (c < '0' || c > '9')
            if (!(c == ',' || c == '.'))
                return false;
    }return true;}
private async void TableButtonRemovePressed(object sender, EventArgs e){
    bool resp = await DisplayAlert(AppResources.RemoveTableTitle, AppResources.RemoveTableBody,
          AppResources.Ok, AppResources.Cancel);
    if (resp){
        int paginaDaRimuovere = this.Children.IndexOf(this.CurrentPage);
        this.Children.RemoveAt(paginaDaRimuovere);
        listaNumeroTavoli.Remove(paginaDaRimuovere + 1);}}
```

Listato 4.5: Code-Behind della pagina `GuestPositionTable.xaml.cs`

MenuCreation

Questa pagina permette all'organizzatore di creare o modificare il menù dell'evento con un reminder che consente di ricontrollare le intolleranze degli invitati e, quindi, di scegliere dei piatti che tengano conto delle stesse. Nel Listato 4.6 abbiamo le parti

fondamentali del codice che gestisce tutta la parte di logica della pagina, mentre nel file `MenuCreation.xaml` vi è la progettazione di tutta la parte grafica della stessa.

```
    //una volta costruita la lista dagli eventi si inizializza la lista degli invitati di ciascun evento
private void SetMenu(IDictionary menu){
    IList Antipastiii = (IList)menu["antipasti"];
    foreach (var item in Antipastiii){
        var Ant = item;
        Antipasti.Add((string)Ant);}
    AntipastiListView.HeightRequest = 50 * Antipasti.Count;
//metodo lanciato al click del frame per aprire il popup
private void IntolerancesPopupOpener_Tapped(object sender, EventArgs e)
{Navigation.PushModalAsync(new GuestsIntolerances(Intolerances));}
private void DishTypePicker_SelectedIndexChanged(object sender, EventArgs e){
    DoneDishesPicker.IsVisible = true;
    var picker = sender as Picker;
//l'elemento scelto verra' aggiunto ad una delle ListView sottostanti, in base al valore del primo picker
private void DoneDishesPicker_SelectedIndexChanged(object sender, EventArgs e){
    var picker = sender as Picker;
    int selectedIndex = picker.SelectedIndex;
    if (selectedIndex != -1){
        var dish = DoneDishesPicker.ItemsSource[selectedIndex] as string;
        switch (DishTypePicker.SelectedIndex){
            case 0:
                if (!Antipasti.Contains(dish)){
                    Antipasti.Add(dish);
                    AntipastiListView.HeightRequest = 50 * Antipasti.Count;}
                DishTypePicker.SelectedIndex = -1;
                DoneDishesPicker.SelectedIndex = -1;
                DoneDishesPicker.IsVisible = false;
                break;
//metodi per aggiungere la portata scritta all'interno della entry nella listview sottostante tramite il
       click del tasto +
private void AntipastiAddButton_Clicked(object sender, EventArgs e){
    var dish = AntipastiEntry.Text;
    if ((dish != "") && (!Antipasti.Contains(dish))){
        Antipasti.Add(dish);
        AntipastiEntry.Text = "";
        AntipastiListView.HeightRequest = 50 * Antipasti.Count;}}
private void PrimiAddButton_Clicked(object sender, EventArgs e){
    var dish = PrimiEntry.Text;
    if ((dish != "") && (!Primi.Contains(dish))){
        Primi.Add(dish);
        PrimiEntry.Text = "";
        PrimiListView.HeightRequest = 50 * Primi.Count;}}
//metodo lanciato al click del bottone crea evento
private async void CreateEventButton_Clicked(object sender, EventArgs e){
    //controllo che sia stata inserita almeno una portata
    if ((Antipasti.Count == 0) && (Primi.Count == 0) && (Secondi.Count == 0) && (Contorni.Count == 0) && (
          Dolci.Count == 0) && (Bevande.Count == 0))
    {DependencyService.Get<IMessage>().ShortAlert(AppResources.EmptyMenu);}
    else{
        RepeatedDishes.Clear();
        //alert di conferma
        bool response;
        if (ModifyEvent){
            response = await DisplayAlert(AppResources.ModifyEvent, AppResources.ModifyEventAlertBody,
                    AppResources.yes, AppResources.no);
        }else{
            response = await DisplayAlert(AppResources.CreateEventAlertTitle, AppResources.
                    CreateEventAlertBody, AppResources.yes, AppResources.no);}
            List<object> invitedContacts = new List<object>();
            List<object> invitedGuests = new List<object>();
            foreach (var guest in GuestList){
            InvitedContacts = invitedContacts;
            InvitedGuests = invitedGuests;
private void GetRepeatedDish(IList currentDishes, IList pastDishes, ObservableCollection<string>
       repeatedDishes, string guestName, string data){
    foreach (string dish in currentDishes){
        if (pastDishes.Contains(dish)){
            string repeatedDish = guestName + ", " + dish + ", " + data;
            if (!repeatedDishes.Contains(repeatedDish))
            {repeatedDishes.Add(repeatedDish);}}}}
//metodo lanciato per tornare alla Mainpage
private async void GoHomeButton_Clicked(object sender, EventArgs e){
    bool response = await DisplayAlert(AppResources.ButtonHomeAlertTitle, AppResources.ButtonHomeAlertBody,
          AppResources.yes, AppResources.no);
    if (response)
    {Application.Current.MainPage = new NavigationPage(new Home.MainPage());}}
```

Listato 4.6: Code-Behind della pagina `MenuCreation.xaml.cs`

4.2.3 Popup

Nella cartella `Popup` sono contenute tutte le pagine che vengono utilizzate come popup personalizzati. Tali pagine vengono di seguito menzionate. Di esse, per motivi di spazio, non viene riportato il codice corrispondente.

GuestIntolerances

Questa pagina viene mostrata quando, nella creazione del menù, vogliamo ricontrollare rapidamente la lista delle intolleranze senza tornare indietro.

ModuloAutocertificazione

Questa pagina viene visualizzata quando si accetta di partecipare ad un evento; infatti, prima di confermare l'invito, si deve compilare un'autocertificazione che attesta che si è in buona salute e si può partecipare all'evento.

PopUpCaricamento

Questa pagina compare una volta completata la registrazione, mentre l'applicazione carica i dati dell'utente del database. Essa simula un caricamento per poi scomparire una volta finito il processo di upload.

RegoleIgieniche

Questa pagina appare quando nella `InfoPage` vogliamo leggere le regole igieniche fornite dal governo.

RegoleNellaRistorazione

Questa pagina appare quando nella `InfoPage` vogliamo leggere le regole per la ristorazione fornite dal governo.

RepeatedDishes

Questa pagina viene visualizzata se, alla fine della creazione del menù, alcuni dei piatti inseriti sono stati già proposti ad alcuni utenti, Essa serve a chiedere un'ulteriore conferma all'utente.

4.2.4 Home

Nella cartella Home sono contenute la `MainPage`, la `SplashPage`, la `EventInfo` e la `InfoPage`. Esse saranno descritte più in dettaglio nelle prossime sottosezioni.

MainPage

Questa è una TabbedPage che ha una scheda per gli eventi attivi, una per gli eventi passati e una per il profilo dell'utente. Nel Listato 4.7 mostriamo le parti fondamentali del codice che gestisce tutta la logica della pagina. Nel Listato 4.8 riportiamo la logica che gestisce il refresh della lista degli eventi, mentre nel file `MainPage.xaml` vi è la progettazione di tutta la parte grafica della pagina.

```
private async void UpdateRegionColor(string ColoreArea){
    String URLString = "http://www.governo.it/it/articolo/domande-frequenti-sulle-misure-adottate-dal-
           governo/15638#zone";
    string HTML;
    using (var wc = new WebClient()){
        HTML = wc.DownloadString(URLString);}
    var doc = new HtmlAgilityPack.HtmlDocument();
    doc.LoadHtml(HTML);
    //HtmlNode element = doc.DocumentNode.SelectSingleNode("//*[contains(normalize-space(text()), " + "'" +
            "area " + ColoreArea + "'" + ")]");
    HtmlNode element = doc.DocumentNode.SelectSingleNode("//*[contains(normalize-space(text()),␣" + "'" + "
          <strong>" + "area␣" + ColoreArea + "</strong>" + "'" + ")]");
    }}}else{
private async void SendNotification(string MyToken){
    string token = MyToken;
    await Task.Run(async () =>{
        //key dell'app che ho recuperato dalla console di firebase
        //FCMClient client = new //FCMClient("AAAAbAO3tsk:
                APA91bGGjqRqfLfSXVEb7dY6YkkoMAL7kbt4KHET10QMbvqIVsL0oMZwoJZvnIs
        //q0l7aoWRWPOMPTNMQu4pY15vsWODPIpht8GTPPkI13BCZvUPOgK6znhSPVbPJkqjJDFeVCuA0qV2S");
        //messaggio da inviare
private async void goToInfoPage(object sender, EventArgs e){
    await Navigation.PushAsync(new InfoPage());}
    //recupero gli id degli eventi
    private async void GetEventsId(){
    var document = await CrossCloudFirestore.Current.Instance.GetCollection("Person").GetDocument(MyId).
           GetDocumentAsync();
        var dataDictionary = document.Data;
        IdEventi = (System.Collections.IList)dataDictionary["idEventi"];
private async void AggiungiAdEventiPassati(string IdEvento){
    var myDocument = await CrossCloudFirestore.Current.Instance.GetCollection("Person").GetDocument(MyId).
           GetDocumentAsync();
    var document = await CrossCloudFirestore.Current.Instance.GetCollection("Event").GetDocument(IdEvento).
           GetDocumentAsync();
    var dataDictionary = document.Data;
    //aggiorno stato dell'invitato se non ha ancora accettato ma l'evento e' passato
    IList DataGuests = (IList)dataDictionary["invitedContacts"];
    IDictionary DataContact;
    for (int i = 0; i < DataGuests.Count; i++){
        DataContact = (IDictionary)DataGuests[i];
        if(MyId.Equals(DataContact["idGuest"].ToString())){
            bool sceltaGuest = Convert.ToBoolean(DataContact["scelta"]);
    dataDictionary["invitedContacts"] = DataGuests;
//recupero i dati degli eventi
private async void GetEventData(){
     EventListView.ItemsSource = EventList;
    for (int i = 0; i < IdEventiRifiutati.Count; i++){
        var document = await CrossCloudFirestore.Current.Instance.GetCollection("Event").GetDocument(
               IdEventiRifiutati[i].ToString()).GetDocumentAsync();
        PastEventListView.ItemsSource = PastEventList;}
private async void SendNotificationForNearlyEvent(string IDEventoNotifica){
    var document = await CrossCloudFirestore.Current.Instance.GetCollection("Event").GetDocument(
           IDEventoNotifica).GetDocumentAsync();
    var dataDictionary = document.Data;
//Al tap di un evento vengono invitati i dati alla pagina che visualizza i dati dell'evento
public async void eventItemTapped(object sender, ItemTappedEventArgs e){
    var itemSelected = e.Item as Event;
    await Navigation.PushAsync(new EventInfo(itemSelected));}
private async void AddPhotoButtonClicked(object sender, EventArgs e){
    await CrossMedia.Current.Initialize();
private async void StoreImage(MediaFile file){
    string uriProfile = MyId + ".jpg";
    var reference = CrossFirebaseStorage.Current.Instance.RootReference.GetChild("profileImages/" +
           uriProfile);
    var metadata = new MetadataChange
    {ContentType = "image/jpeg"};
private async void ModifyIntol(object sender, EventArgs e){
    choice = true;
    await Navigation.PushAsync(new AddYourIntolerances(choice, phoneNumber, fileImage, regionSelected));}
private void LogOut(object sender, EventArgs e){
    var interfacesss = DependencyService.Get<IAuthFirebase>();
    interfacesss.LogOutFirebaseEmail();
    Application.Current.MainPage = new NavigationPage(new FirstChoice());
    DependencyService.Get<IMessage>().ShortAlert(AppResources.SeeYouSoon);}
```

Listato 4.7: Code-Behind della pagina `MainPage.xaml.cs`

```
public ICommand RefreshCommand => new Command(async () => await RefreshItemsAsync());
public MainPageViewModel(){}
//Indicatore per il refresh della lista degli eventi
async Task RefreshItemsAsync(){
    IsRefreshing = true;
    await Task.Delay(TimeSpan.FromSeconds(RefreshDuration));
    Application.Current.MainPage = new NavigationPage(new MainPage());
    IsRefreshing = false;}
#region INotifyPropertyChanged
public event PropertyChangedEventHandler PropertyChanged;
void OnPropertyChanged([CallerMemberName] string propertyName = null)
{PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));}
#endregion}}
```

Listato 4.8: ViewModel della pagina `MainPageViewModel.cs`

SplashPage

Questa pagina viene presentata ogni volta all'avvio dell'applicazione. Essa mostra il nome e il logo dell'app. Inoltre, in essa vi è anche un controllo sulla connettività necessaria per prelevare i dati dal database Firestore, nonchè una verifica di primo accesso nell'applicazione.

EventInfo

In questa pagina l'invitato può trovare tutte le informazioni dell'evento (data, ora, menù, etc.) a cui è stato invitato e può decidere se parteciparvi o rifiutare l'invito. Nel Listato 4.9 abbiamo le parti di codice che gestisce tutta la logica della pagina, mentre, nel file `EventInfo.xaml`, vi è la progettazione di tutta la parte grafica della pagina.

```
//ottengo i dati degli invitati dal DB
private void getGuestsData(IList invitatiIscritti, IList invitatiNonIscritti){
       IDictionary Invitato;
       for (int i = 0; i < invitatiIscritti.Count; i++){
           Invitato = (IDictionary)invitatiIscritti[i];
           getData(Invitato);}
       IDictionary Invitatononiscritto;
       for (int i = 0; i < invitatiNonIscritti.Count; i++){
           Invitatononiscritto = (IDictionary)invitatiNonIscritti[i];
           Guest guest = new Guest{
               Name = Invitatononiscritto["nameGuest"].ToString(),
               Phone = Invitatononiscritto["phoneGuest"].ToString(),
               Email = Invitatononiscritto["emailGuest"].ToString(),
               Accettato = true,
               Scelta = false,
               UriProfile = "ic_person_bot_add_18dp.xml"};
           GuestContactList.Add(guest);
           AllContacts.Add(guest);}}
   //metodo per mostrare il menu delle opzioni all'utente che ha rifiutato
   private void ShowDeclineMenu(){
       ToolbarItem item = new ToolbarItem{
           Text = AppResources.DeleteFromTheList,
           Order = ToolbarItemOrder.Secondary,
           Priority = 0,};
       item.Clicked += DeleteStoryEvent;
       this.ToolbarItems.Add(item);}
   //metodo per mostrare il menu delle opzioni all'invitato
   private void ShowGuestMenu(bool scelta){
       this.ToolbarItems.Clear();
       if (!scelta){
           ToolbarItem itemAccept = new ToolbarItem{
               Order = ToolbarItemOrder.Primary,
               Priority = 0,
               IconImageSource = ImageSource.FromFile("accept.xml")};
           itemAccept.Clicked += AcceptInvite;
           this.ToolbarItems.Add(itemAccept);
           ToolbarItem itemDecline = new ToolbarItem{
               Order = ToolbarItemOrder.Primary,
               StyleId = "accept_and_decline",
               Priority = 0,
               IconImageSource = ImageSource.FromFile("decline.xml")};
           itemDecline.Clicked += DeclineInvite;
```

```
            this.ToolbarItems.Add(itemDecline);
        }else{
            ToolbarItem item = new ToolbarItem{
                Text = AppResources.LeaveEvent,
                Order = ToolbarItemOrder.Secondary,
                Priority = 0,};
            item.Clicked += DeclineInvite;
            this.ToolbarItems.Add(item);}}
    //metodo per mostrare il menu delle opzioni dell'organizzatore
    private void ShowOrganizerMenu(){
        ToolbarItem item = new ToolbarItem{
            Text = AppResources.DeleteEvent,
            Order = ToolbarItemOrder.Secondary,
            Priority = 0};
        item.Clicked += RemoveEvent;
        this.ToolbarItems.Add(item);
        ToolbarItem item2 = new ToolbarItem{
            Text = AppResources.ModifyEvent,
            Order = ToolbarItemOrder.Secondary,
            Priority = 0};
        item2.Clicked += ModifyEvent;
        this.ToolbarItems.Add(item2);}
    //caso in cui si preme sul pulsante accetta
    private async void AcceptInvite(object sender, EventArgs e){
        bool response = await DisplayAlert(AppResources.Confirm, AppResources.AcceptInvitation, AppResources.
            yes, AppResources.no);
        if (response)
        {Navigation.PushModalAsync(new ModuloAutocertificazione(eventoDaMandare));
            //Title = Titolo;
            //UpdateGuestData(true);}}
    private async void RemoveEvent(object sender, EventArgs e){
        bool response = await DisplayAlert(AppResources.RemoveEventAlertTitle, AppResources.
            RemoveEventAlertBody, AppResources.yes, AppResources.no);
            var dataDictionary = document.Data;
            IList DataGuests = (IList)dataDictionary["invitedContacts"];
            IDictionary DataContact;
            IList Eventi;
            IList EventiPassati;
            for (int i = 0; i < DataGuests.Count; i++){
                DataContact = (IDictionary)DataGuests[i];
                var doc = await CrossCloudFirestore.Current.Instance.GetCollection("Person").GetDocument(
                      DataContact["idGuest"].ToString()).GetDocumentAsync();
                var datiPerson = doc.Data;
                Eventi = (IList)datiPerson["idEventi"];
                EventiPassati = (IList)datiPerson["idEventiPassati"];
                Eventi.Remove(IdEvento);
                if (EventiPassati.Contains(IdEvento)) EventiPassati.Remove(IdEvento);
                datiPerson["idEventi"] = Eventi;
                datiPerson["idEventiPassati"] = EventiPassati;
            await Navigation.PopAsync();
            Application.Current.MainPage = new NavigationPage(new MainPage());
            DependencyService.Get<IMessage>().ShortAlert(AppResources.DeleteEventToast);}}
    private async void ModifyEvent(object sender, EventArgs e)
    {await Navigation.PushAsync(new NewEvent(true, eventoDaMandare));}
```

Listato 4.9: Code-Behind della pagina `EventInfo.xaml.cs`

InfoPage

Questa pagina è dedicata alle informazioni relative alla pandemia globale da Covid-19, come, ad esempio, le regole igieniche, i colori delle regioni o i numeri dei contagi regionali o nazionali.

4.2.5 Interfaces

La cartella `Interfaces` contiene le interfacce che servono per visualizzare i toast e per cambiare il tema dell'applicazione. Nel seguito, per ragioni di spazio non mostreremo in dettaglio il codice di tali interfacce.

IMessage

Questa interfaccia serve ad implementare la visualizzazione di toast.

ITheme

Questa interfaccia serve ad implementare il cambio da tema chiaro a tema scuro.

4.2.6 LoginAndSignin

Questa cartella contiene tutte le pagine utili per l'accesso o la creazione di un nuovo account per la nostra applicazione. Nel seguito, per ragioni di spazio, mostreremo in dettaglio il codice solo di alcune pagine.

FirstChoice

In questa pagina l'utente sceglie se effettuare il login, e quindi entrare nell'applicazione come utente già registrato, oppure creare un nuovo profilo, e quindi procedere con il percorso di registrazione.

AccountChoice

Questa pagina viene utilizzata per scegliere quale tipologia di account si vuole utilizzare per registrarsi o per accedere nell'applicazione (Google, Facebook o email).

PhotoChoice

Questa pagina viene utilizzata dagli utenti che hanno scelto di registrarsi nella nostra applicazione, per inserire il numero di telefono e l'immagine di profilo. Nel Listato 4.10 mostriamo il codice che gestisce tutta la logica della pagina, mentre nel file `PhotoChoice.xaml` vi è la progettazione di tutta la parte grafica della stessa.

```
//Questo metodo viene richiamato sia dall'image button sia dal button normale e servira' per aggiungere un
       immagine del profilo
private async void AddPhotoButtonClicked(object sender, EventArgs e){
    await CrossMedia.Current.Initialize();
    try{
        //Tramite PickPhotoAsync possiamo prendere una foto dalla nostra galleria e ne settiamo la
               dimensione della risoluzione
        file = await Plugin.Media.CrossMedia.Current.PickPhotoAsync(new Plugin.Media.Abstractions.
               PickMediaOptions{
            PhotoSize = Plugin.Media.Abstractions.PhotoSize.Medium});
        //Se non abbimo scelto nulla ritorna nulla
        if (file == null) return;
        //Se quindi e' stata scelta invece una foto possiao aggiungerla all'ImageButton e ritorniamo lo
               stream di quell'immagine
        photo_add.Source = ImageSource.FromStream(() =>{
            var imageStram = file.GetStream();
            return imageStram;});
        photo_add.Padding = 0; //Settiamo il padding a 0 in modo che aderisca al meglio all'imageButton}
    catch (Exception ex)
    {Debug.WriteLine(ex.Message);}}
private async void GoNextButtonClicked(object sender, EventArgs e){
    phoneNumber = phoneNumberTxt.Text;
    if (phoneNumber != null) //Il phoneNumber e' un campo obbligatorio e quindi nel caso non ci fosse non
           va avanti{
        //Passiamo al successivo step della registrazione e gli passiamo:
        // - logOrSign(che dovremo poi passare all'email)
        // - phoneNumber
        // - il file che si abbia scelta o meno la foto del profilo
        await Navigation.PushAsync(new ChooseRegion(logOrSing, phoneNumber, file));}
    else{DependencyService.Get<IMessage>().ShortAlert(AppResources.InsertPhoneToast);}}
//Se si preme il button back si ritorna alla prima scelta cancellando tutti i dati gia' inseriti quindi per
       sicurezza si chiede conferma tramite un Alert
private async void BackButtonClicked(object sender, EventArgs e){
    bool response = await DisplayAlert(AppResources.AttenctionAlertTitle, AppResources.AttenctionAlertBody,
           AppResources.yes, AppResources.no);
    if (response)
    {Application.Current.MainPage = new NavigationPage(new FirstChoice());}}
```

Listato 4.10: Code-Behind della pagina `PhotoChoice.xaml.cs`

ChooseRegion

Questa pagina viene utilizzata per scegliere la regione in cui l'utente si trova attualmente. Nel Listato 4.11 mostriamo il codice che gestisce tutta la logica della pagina, mentre nel file `ChooseRegion.xaml` vi è la progettazione di tutta la parte grafica della stessa.

```
private async void SetupPicker(){
    var orderedList = new List<string>();
    orderedList.Add("Abruzzo");
    orderedList.Add("Basilicata");
    orderedList.Add("Calabria");
    orderedList.Add("Campania");
    orderedList.Add("Emilia␣Romagna");
    orderedList.Add("Friuli␣Venezia␣Giulia");
    orderedList.Add("Lazio");
    orderedList.Add("Liguria");
    orderedList.Add("Lombardia");
    orderedList.Add("Marche");
    orderedList.Add("Molise");
    orderedList.Add("Piemonte");
    orderedList.Add("Provincia␣Autonoma␣di␣Bolzano");
    orderedList.Add("Provincia␣autonoma␣di␣Trento");
    orderedList.Add("Puglia");
    orderedList.Add("Sardegna");
    orderedList.Add("Sicilia");
    orderedList.Add("Toscana");
    orderedList.Add("Valle␣d'Aosta");
    orderedList.Add("Veneto");
    orderedList.Add("Umbria");
    RegionsPicker.ItemsSource = orderedList;}
private void RegionsePicker_SelectedIndexChanged(object sender, EventArgs e){
    var picker = sender as Picker;
    int selectedIndex = picker.SelectedIndex;
    if (selectedIndex != -1)
    {RegionSelected = ((string)RegionsPicker.ItemsSource[selectedIndex]);}}
private async void GoNextButtonClicked(object sender, EventArgs e){
    if (!(string.IsNullOrEmpty(RegionSelected)))
    {await Navigation.PushAsync(new AddYourIntolerances(ModifyOrSing, phone, file, RegionSelected));}
    else{DependencyService.Get<IMessage>().ShortAlert(AppResources.SelectRegionToast);}}
//Se si preme il button back si ritorna alla prima scelta cancellando tutti i dati gia' inseriti quindi per
        sicurezza si chiede conferma tramite un Alert
private async void BackButtonClicked(object sender, EventArgs e){
    bool response = await DisplayAlert(AppResources.AttenctionAlertTitle, AppResources.AttenctionAlertBody,
            AppResources.yes, AppResources.no);
    if (response)
    {Application.Current.MainPage = new NavigationPage(new FirstChoice());}}}}
```

Listato 4.11: Code-Behind della pagina `ChooseRegion.xaml.cs`

AddIntolerances

In questa pagina l'utente seleziona le sue intolleranze (nel caso non ne avesse, lascerà vuoto tale campo). Nel Listato 4.12 mostriamo il codice che gestisce tutta la parte di logica della pagina, mentre nel file `AddIntolerances.xaml` vi è la progettazione di tutta la parte grafica della stessa.

```
//metodo per impostare i dati del picker delle Intolleranze prendendoli dal db
private async void SetupPicker(){
    var doc = await CrossCloudFirestore.Current.Instance.GetCollection("Utilities").GetDocument("
           Intolleranze").GetDocumentAsync();
    var list = (IList)doc.Data["Intolleranze"]; //Si va ad estrarre tutte le intolleranze dal db
    var orderedList = new List<string>();
    foreach (string item in list){
        orderedList.Add(item);   //Si va a prendere ogni elemento estratto dal doc intolleranze e si va ad
              inserire in una lista}
    orderedList.Sort();
    IntolerancePicker.ItemsSource = orderedList; //E si vanno ad assegnare poi al picker}
//metodo per inserire le intolleranze dell'utente corrente nella ListView prendendole dal db
private async void SetupYourIntolModify(){
    var getUid = DependencyService.Get<IAuthFirebase>(); //Si va inanzitutto a prendere l'uid dell'utente
          corrente che corrisponde al nome del suo doc nel db
    string uid = getUid.GetUserUid();
    var doc = await CrossCloudFirestore.Current.Instance.GetCollection("Person").GetDocument(uid).
           GetDocumentAsync();
    var intoleranceList = (IList)doc.Data["intolleranze"]; //E poi si va a estrarre il documento dal db e
           si va a prendere il campo intolleranze
```

```
    foreach (var item in intoleranceList){
        var intolerance = new Intolerance((string)item);
        GuestIntolerances.Add(intolerance);     //Qui si vanno a inserire in 2 liste una per la
                visualizzazione tramite listView
        intol.Add(intolerance.Name);            //E una con solo i nomi delle intolleranze che serve poi per
                 caricarla nel db }
    intol.Sort();
    IntolerancesListView.ItemsSource = GuestIntolerances;}
//metodo lanciato quando si seleziona uno degli elementi del picker in cui l'elemento selezionato viene
       aggiunto all'ItemsSource della ListView
private void IntolerancePicker_SelectedIndexChanged(object sender, EventArgs e){
    var picker = sender as Picker;
    int selectedIndex = picker.SelectedIndex;
    if (selectedIndex != -1){
        var intolerance = new Intolerance((string)IntolerancePicker.ItemsSource[selectedIndex]);
        if (!GuestIntolerances.Contains(intolerance))  //Controlliamo se l'intolleranza selezionata e' gia'
               presente nella lista e se non lo e' la aggiungiamo{
            GuestIntolerances.Add(intolerance);
            intol.Add(intolerance.Name);}}}
//metodo lanciato al click per eleminare l'intolleranza selezionata
private async void DeleteButton_Clicked(object sender, EventArgs e){
    bool response = await DisplayAlert(AppResources.DeleteIntolerancesTitle, AppResources.
           DeleteIntolerancesBody, AppResources.yes, AppResources.no);
    if (response){
        var button = sender as ImageButton;
        var intolerance = button.BindingContext as Intolerance;
        GuestIntolerances.Remove(intolerance);
        intol.Remove(intolerance.Name);}}
private async void GoNextButtonClicked(object sender, EventArgs e){
    if (ModifyOrSing) //Se e' true e quindi si stava modificando le intolleranze gia' presenti, si va a
           fare l'update del campo intolleranze nel bd{
        var getUid = DependencyService.Get<IAuthFirebase>();
        string uid = getUid.GetUserUid();
        await CrossCloudFirestore.Current
                .Instance
                .GetCollection("Person")
                .GetDocument(uid)
                .UpdateDataAsync(new { intolleranze = intol });
        Application.Current.MainPage = new NavigationPage(new Home.MainPage());
    }else   //Altrimenti si va a completare la registrazione nell'emailAuth passandogli tutti i dati
           precedentemente inseriti e le intolleranze
    {await Navigation.PushAsync(new EmailAuth(ModifyOrSing, intol, phone, file, regioneSelected));}}
//Se si preme il button back si ritorna alla prima scelta cancellando tutti i dati gia' inseriti quindi per
        sicurezza si chiede conferma tramite un Alert
private async void BackButtonClicked(object sender, EventArgs e){
    bool response = await DisplayAlert(AppResources.AttenctionAlertTitle, AppResources.AttenctionAlertBody,
            AppResources.yes, AppResources.no);
    if (response)
    {Application.Current.MainPage = new NavigationPage(new FirstChoice());}}}}
```

Listato 4.12: Code-Behind della pagina `AddYourIntolerances.xaml.cs`

EmailAuth

Questa pagina viene utilizzata per effettuare l'accesso tramite email o per concludere la registrazione. Nel Listato 4.13 visualizziamo il codice che gestisce tutta la logica della pagina, mentre nel file `EmailAuth.xaml` vi è la progettazione di tutta la parte grafica della stessa.

```
private async void GoNextButtonClicked(object senders, EventArgs es){
    email = emailTxt.Text;
    password = passwordTxt.Text;
    name = nameTxt.Text;
    //Quando si preme il pulsante per andare avanti e completare la registrazione o fare il login
    //si fanno diversi controlli sui campi nome e cognome (solo per signin), email e password
    if (string.IsNullOrEmpty(email)) DependencyService.Get<IMessage>().ShortAlert(AppResources.EmailMiss);
    else if (!email.Contains("@") || !email.Contains(".")) DependencyService.Get<IMessage>().ShortAlert(
           AppResources.EmailNotValid);
    else if (string.IsNullOrEmpty(password)) DependencyService.Get<IMessage>().ShortAlert(AppResources.
           PasswordMiss);
    else if (password.Length < 6) DependencyService.Get<IMessage>().ShortAlert(AppResources.PasswordShort);
    else if (string.IsNullOrEmpty(name) && !(logOrSing)) DependencyService.Get<IMessage>().ShortAlert(
           AppResources.NameMissing);
    else{if (logOrSing) //Va qui se ha cliccato login e quindi logOrSign = true{
           var emailLogin = DependencyService.Get<IAuthFirebase>();
           string token = await emailLogin.LoginWithEmailPassword(email, password);
           if (token.Equals("0")) //Se l'account che ha messo l'utente per il LOGIN non esiste nel db di
                  Firebase
           {DependencyService.Get<IMessage>().ShortAlert(AppResources.AccountNotFound);}
           else if (token.Equals("1")) //Se c'e' un qualche tipo di errore differente dal precedente
           {DependencyService.Get<IMessage>().ShortAlert(AppResources.ErrorAccour);}
           else    //Se l'account esiste e l'utente vuole fare il LOGIN lo fa accedere
```

```
                {await Navigation.PushModalAsync(new PopUpCaricamento());
                    DependencyService.Get<IMessage>().ShortAlert(AppResources.AccessDone);
                    Application.Current.MainPage = new NavigationPage(new Home.MainPage());}
            }else //Va qui se ha cliccato registrati e quindi logOrSign = false{
                var emailLogin = DependencyService.Get<IAuthFirebase>();
                string token = await emailLogin.SigninWithEmailPassword(email, password);
                string uid = emailLogin.GetUserUid();
                if (token.Equals("0")) //Se l'account che ha messo l'utente per il SIGNIN esiste nel db di
                       Firebase ma lui vuole registrarsi di nuovo con lo stesso
                {DependencyService.Get<IMessage>().ShortAlert(AppResources.EmailJustRegistrated);}
                else      //Se l'utente vuole fare il SIGNIN gli fa creare l'account
                {await Navigation.PushModalAsync(new PopUpCaricamento());
                    if (file != null) //Se l'utente ha scelto una foto precedentemente{
                        uriProfile = uid + ".jpg"; //Diamo come nome dell'immagine l'UID perche' diverso per ogni
                                utente
                        var reference = CrossFirebaseStorage.Current.Instance.RootReference.GetChild("
                               profileImages/" + uriProfile);
                        var metadata = new MetadataChange
                        {ContentType = "image/jpeg"};
                        await reference.PutStreamAsync(file.GetStream(), metadata);}
                    await CrossCloudFirestore.Current //Genera il documento associato all'uid dell'utente
                     .Instance
                     .GetCollection("Person")
                     .GetDocument(uid)
                     .SetDataAsync(
                        new{email,
                            name = name,
                            phone = phone,
                            region = regionSelected,
                            token = token,
                            malato = false,
                            privato = false,
                            uriProfile = uriProfile,
                            idEventi = emptyList,
                            idEventiPassati = emptyList,
                            intolleranze = intol,
                            dishes = emptyDish,
                            friends = emptyList,
                            friendsRequestsDone = emptyList,
                            friendsRequestsReceived = emptyList});
                    DependencyService.Get<IMessage>().ShortAlert(AppResources.AccountCreated);
                    Application.Current.MainPage = new NavigationPage(new Home.MainPage());}}}}
    //Diversamente dagli altri back button qui si fa un controllo in piu' perche' se l'utente ha cliccato su
           login
    //non serve andare a mostrare l'alert che indica il rischio di cancellazione dei dati gia' inseriti perche'
            appunto non abbiamo inserito ancora nulla
    private async void BackButtonClicked(object sender, EventArgs e){
        if (logOrSing)
        {Application.Current.MainPage = new NavigationPage(new FirstChoice());}
        else{
            bool response = await DisplayAlert(AppResources.AttenctionAlertTitle, AppResources.
                   AttenctionAlertBody, AppResources.yes, AppResources.no);
            if (response)
            {Application.Current.MainPage = new NavigationPage(new FirstChoice());}}}}
```

Listato 4.13: Code-Behind della pagina `EmailAuth.xaml.cs`

IAuthFirebase

Questa interfaccia serve ad implementare tutta la parte di autenticazione di Firebase.

4.2.7 Models

In questa cartella possiamo trovare una serie di modelli utilizzati per la realizzazione dell'app. Nel seguito citiamo tali modelli ma non presentiamo il codice corrispondente per ragioni di spazio.

Event

Il file `Event.cs` rappresenta il modello per descrivere la struttura di un evento.

Guest

Il file `Guest.cs` rappresenta il modello per descrivere la struttura di un invitato bot.

Intolerance

Il file `Intolerance.cs` rappresenta il modello per descrivere la struttura di un'intolleranza.

Person

Il file `Person.cs` rappresenta il modello per descrivere la struttura di un invitato.

TableGuest

Il file `TableGuest.cs` rappresenta il modello per descrivere la struttura di un invitato da posizionare intorno al tavolo.

4.2.8 Resx

In questa cartella ci sono i file contententi tutte le stringhe tradotte in una determinata lingua. Quest'ultima viene specificata nel nome del file. I file contenuti in questa cartella sono i seguenti:

- `AppResources.en-US.resx`: questo file rappresenta tutte le stringhe tradotte nella lingua inglese.
- `AppResources.it.resx`: questo file rappresenta tutte le stringhe tradotte nella lingua italiana.
- `AppResources.resx`: questo file rappresenta tutte le stringhe tradotte nella lingua inglese che usiamo come lingua di default, e che, quindi, verranno utilizzate nel caso in cui il dispositivo in cui è installata l'applicazione sia configurato con una lingua non presente nei file precendeti. Per ragioni di spazio non possiamo mostrare il codice relativo a tali file.

4.2.9 Theme

In questa cartella sono presenti le risorse che servono per dichiarare i colori per la Light Mode e per la Dark Mode. Più specificatamente:

- Nel file `DarkTheme.xaml` vi è tutta la lista delle componenti del `ResourceDictionary` che servono per definire il tema scuro.
- Nel file `LightTheme.xaml` vi è tutta la lista delle componenti del `ResourceDictionary` che servono per definire il tema chiaro.

Per ragioni di spazio non possiamo mostrare il codice relativo a tali file.

4.3 HUP.Android

La sezione `HUP.Android` è un progetto specifico della piattaforma contenente il codice nativo e le risorse come immagini e colori.

4.3.1 AndroidManifest

Il manifest contiene le caratteristiche principali dell'applicazione, senza le quali non sarebbe possibile compilare alcuna porzione di codice. In esso vanno inseriti i permessi che servono all'applicazione per funzionare, il nome, l'icona e il tema.

4.3.2 MainActivity

Questo file rappresenta la MainActivity dell'applicazione Android dove sono stati dichiarati i metodi per chiedere i permessi di cui essa necessita per poter funzionare. In questo file, inoltre, sono stati settati i flag per l'utilizzo di funzionalità ancora in sviluppo, come il Drag&Drop nativo di Xamarin.

4.3.3 Google Services

Il file `google-services.json` ci viene fornito durante il procedimento di configurazione della nostra applicazione con il database di Firebase.

4.3.4 Drawable

Nella cartella `Drawable` sono contenute tutte le immagini (immagini JPG, PNG, vettoriali, etc.) utilizzate all'interno dell'applicazione.

4.3.5 Mipmap

Nella cartella `Mipmap` è contenuta l'immagine del logo dell'applicazione nelle diverse misure che vengono utilizzate in base alla versione del sistema operativo e alla grandezza dello schermo.

4.3.6 Value

Nella cartella `Value` sono contenuti i dizionari degli stili e dei colori. Più specificatamente:

- Nel file `color.xml` abbiamo l'elenco dei colori utilizzati all'interno dell'applicazione.
- Nel file `style.xml` abbiamo l'elenco degli stili utilizzati all'interno dell'applicazione.

4.3.7 Services

In questa cartella viene configurato il service per la ricezione delle notifiche. Il codice corrispondente a tale service viene riportato nel Listato 4.14

```
        public override void OnMessageReceived(RemoteMessage message){
            if (Build.VERSION.SdkInt >= Android.OS.BuildVersionCodes.O){
                FirebasePushNotificationManager.DefaultNotificationChannelId = "FirebasePushNotificationChannel";
                FirebasePushNotificationManager.DefaultNotificationChannelName = "General";
                FirebasePushNotificationManager.DefaultNotificationChannelImportance = NotificationImportance.High;}
#if DEBUG
            FirebasePushNotificationManager.Initialize(Android.App.Application.Context, true);
#else
            FirebasePushNotificationManager.Initialize(Android.App.Application.Context, false);
#endif
            var intent = new Intent(Android.App.Application.Context, typeof(MainActivity));
            intent.AddFlags(ActivityFlags.ClearTop);
            var pendingIntent = PendingIntent.GetActivity(Android.App.Application.Context, 1234, intent,
                   PendingIntentFlags.OneShot);
            NotificationCompat.BigTextStyle textStyle = new NotificationCompat.BigTextStyle();
            textStyle.BigText(message.Data["body"]);
            var notificationBuilder = new NotificationCompat.Builder(Android.App.Application.Context, "
                   FirebasePushNotificationChannel")
                                                  .SetContentTitle(message.Data["title"])
                                                  .SetContentText(message.Data["body"])
                                                  .SetContentIntent(pendingIntent)
                                                  .SetAutoCancel(true) .SetLargeIcon(BitmapFactory.
                                                         DecodeResource(Android.App.Application.Context.
                                                         Resources, Resource.Drawable.Logo))
                                                  .SetSmallIcon(Resource.Drawable.Logo)
                                                  .SetStyle(textStyle);
            var notificationManager = NotificationManagerCompat.From(Android.App.Application.Context);
            notificationManager.Notify(1234, notificationBuilder.Build());
            base.OnMessageReceived(message);}
        public override void OnNewToken(string tokenReceived){
            var getUid = Xamarin.Forms.DependencyService.Get<LoginAndSignin.IAuthFirebase>();
            if(getUid != null){
                count = Preferences.Get("countInitial", 0);
                if(count == 0){count++;
                    Preferences.Set("countInitial", count);}else{
                    string uid = getUid.GetUserUid();
                    CrossCloudFirestore.Current
                     .Instance
                     .GetCollection("Person")
                     .GetDocument(uid)
                     .UpdateDataAsync(new { token = tokenReceived });
                    base.OnNewToken(tokenReceived);}}}}}
```

Listato 4.14: Codice del service `NotificationService.cs`

4.3.8 Interfaces

In questa cartella abbiamo il codice delle classi che implementano le interfacce in Android. In particolare:

- Nel Listato 4.15 troviamo il codice che implementa l'interfaccia `MessageAndroid`, per i toast in Android.

```
        public class MessageAndroid : IMessage{
            public void LongAlert(string message)
            {Toast.MakeText(Application.Context, message, ToastLength.Long).Show();}
            public void ShortAlert(string message)
            {Toast.MakeText(Application.Context, message, ToastLength.Short).Show();}}}
```

Listato 4.15: Codice della classe `MessageAndroid.cs`

- Nel Listato 4.16 troviamo il codice che implementa l'interfaccia `AndroidAuth`, per l'autenticazione con Firebase.

```
        public class AndroidAuth : IAuthFirebase{
            public bool IsUserLoggedIn() => FirebaseAuth.Instance.CurrentUser != null;
            public async Task<string> LoginWithEmailPassword(string email, string password){
```

```
        try{
            var user = await FirebaseAuth.Instance.SignInWithEmailAndPasswordAsync(email, password);
            var token = await user.User.GetIdTokenAsync(false);
            return token.Token;}
        catch (FirebaseAuthInvalidUserException notFound)
        {return "0";}
        catch (Exception e)
        {return "1";}}
    public async Task<string> SigninWithEmailPassword(string email, string password){try{
            var user = await FirebaseAuth.Instance.CreateUserWithEmailAndPasswordAsync(email, password);
            var token = await user.User.GetIdTokenAsync(false);
            return token.Token;}
        catch (Exception e)
        {return "0";}}
    public void LogOutFirebaseEmail()
    {FirebaseAuth.Instance.SignOut();}
    public string GetUserUid()
    {return FirebaseAuth.Instance.CurrentUser.Uid;}
    public async Task<string> GetToken()
    {var instanceIdResult = await FirebaseInstanceId.Instance.GetInstanceId().AsAsync<IInstanceIdResult
            >();
        return instanceIdResult.Token;}}}
```

Listato 4.16: Codice della classe `AndroidAuth.cs`

4.4 Pacchetti NuGet

I Pacchetti NuGet sono delle librerie aggiuntive create dalla community di Xamarin che servono ad implementare nuove funzionalità all'interno della nostra applicazione. In particolare, i pacchetti da noi utilizzati sono i seguenti:

- `FirebaseNet16`: Questo pacchetto viene utilizzato per inviare messaggi di notifica utilizzando i server di connessione HTTP FCM.
- `HtmlAgilityPack`: Questo pacchetto viene utilizzato per collegare l'app al sito del governo, dove vi sono tutte le informazioni sulle nuove normative anti Covid-19, e per ottenere i colori delle regioni aggiornati in tempo reale, sempre dallo stesso sito.
- `Plugin.CloudFirestore`: Questo pacchetto viene utilizzato per l'implementazione specifica della sezione Database Firestore di Firebase, che viene utilizzato per memorizzare tutti i dati principali degli utenti e degli eventi.
- `Plugin.FirebasePushNotification`: Questo pacchetto viene utilizzato per ricevere e gestire le push notification di Firebase nelle piattaforme Android e iOS.
- `Plugin.FirebaseStorage`: Questo pacchetto viene utilizzato per l'implementazione specifica della sezione Storage di Firebase, che noi utilizziamo per memorizzare le immagini del profilo degli utenti.
- `Plugin.Share`: Questo pacchetto interagisce con `HtmlAgilityPack` e implementa le funzioni di apertura delle pagine browser tramite link.
- `Xam.Plugin.Connettivity`: Questo pacchetto viene utilizzato per implementare il riconoscimento di connessione a Internet e per specificare se essa è disponibile o meno dal momento che la nostra applicazione necessita, appunto, di prelevare tutte le informazioni necessarie da Firebase.
- `Xam.Plugin.Media`: Questo pacchetto viene utilizzato per scattare foto o per sceglierne una dalla galleria, per poi impostarla come immaine del profilo dell'utente.

- `Xam.Plugins.Notifier`: Questo pacchetto viene utilizzato per generare messaggi di notifica dopo che questi sono stati raccolti tramite le funzioni messe a disposizione dal pacchetto `Plugin.FirebasePushNotification`.
- `Xamarin.Essentials`: Questo pacchetto offre agli sviluppatori delle API essenziali multipiattaforma per le loro applicazioni mobile. iOS, Android e UWP offrono API uniche per la loro piattaforma e per il loro sistema operativo a cui gli sviluppatori hanno accesso, il tutto in C# sfruttando Xamarin. Ciò significa che gli sviluppatori devono imparare tre diverse API solo per accedere alle funzionalità specifiche della piattaforma. Con Xamarin.Essentials, gli sviluppatori dispongono di un'unica API multipiattaforma che funziona con qualsiasi applicazione iOS, Android o UWP, a cui è possibile aggiungere del codice condiviso, indipendentemente dalla modalità di creazione dell'interfaccia utente.
- `Xamarin.Firebase.Auth`: Questo pacchetto viene utilizzato per l'implementazione specifica della sezione Autenticazione di Firebase.
- `Xamarin.Firebase.Core`: Questo pacchetto viene utilizzato per l'implementazione di tutto il sistema di Firebase.

5

Manuale Utente

In questo capitolo verranno mostrate le linee guida per l'utilizzo dell'applicazione la cui implementazione è stata descritta nei capitoli precedenti. Si vedranno tutti i passaggi da compiere per utilizzare l'app, facendo riferimento agli screen-shot dell'applicativo, derivati dal dispositivo su cui essa è stata implementata.

5.1 Registrazione

Dopo aver installato l'applicazione sul dispositivo, l'utente potrà accedervi ad essa premendo sull'icona corrispondente, presente nella schermata delle applicazioni dello smartphone (Figura 5.1).

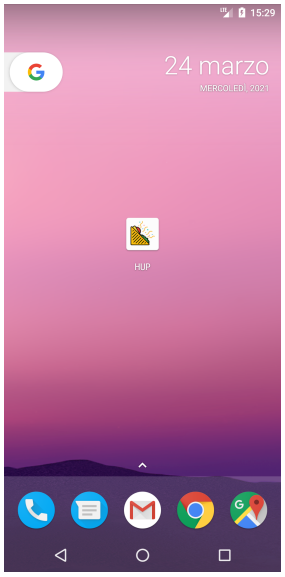


Figura 5.1: Icona visibile dal menù del dispositivo

In seguito all'apertura, all'utente verrà presentata una Splash page di pochi secondi, che mostrerà il logo e il nome dell'app (Figura 5.2).

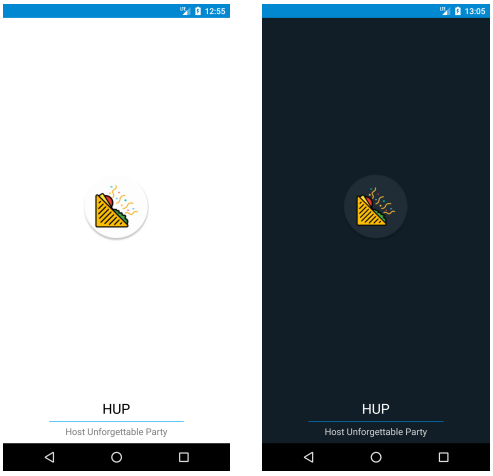


Figura 5.2: SplashPage iniziale in LightMode e in DarkMode

In essa, inoltre, abbiamo un controllo per la connessione ad Internet (Figura 5.3), fondamentale per prendere i dati da Firebase, e un controllo sull'utente per verificare se ce ne sia uno già connesso nel dispositivo. In particolare:

- se l'utente è già connesso andiamo direttamente alla MainPage;
- se non lo è andremo nella pagina FirstChoice;

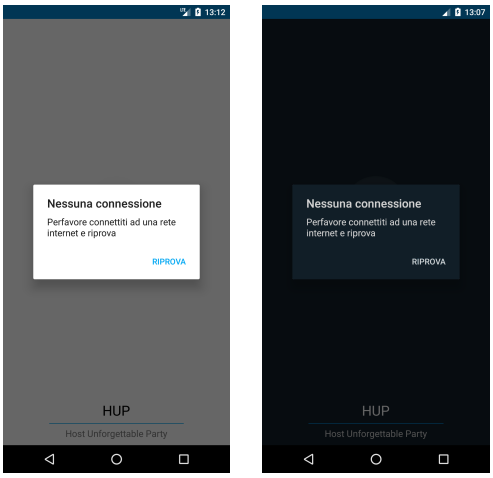


Figura 5.3: Connessione assente in LightMode e in DarkMode

Si è pensato di realizzare per tutte le pagine della registrazione, un'interfaccia più pulita, senza appbar superiore, ma che contenesse il logo e il nome dell'app.

Se è la prima volta che apriamo l'applicazione ci troveremo davanti una pagina che ha al proprio interno 2 pulsanti (Figura 5.4). Essi consentono di entrare nell'applicazione come nuovo utente o come utente già esistente (se si è già registrati in precedenza).

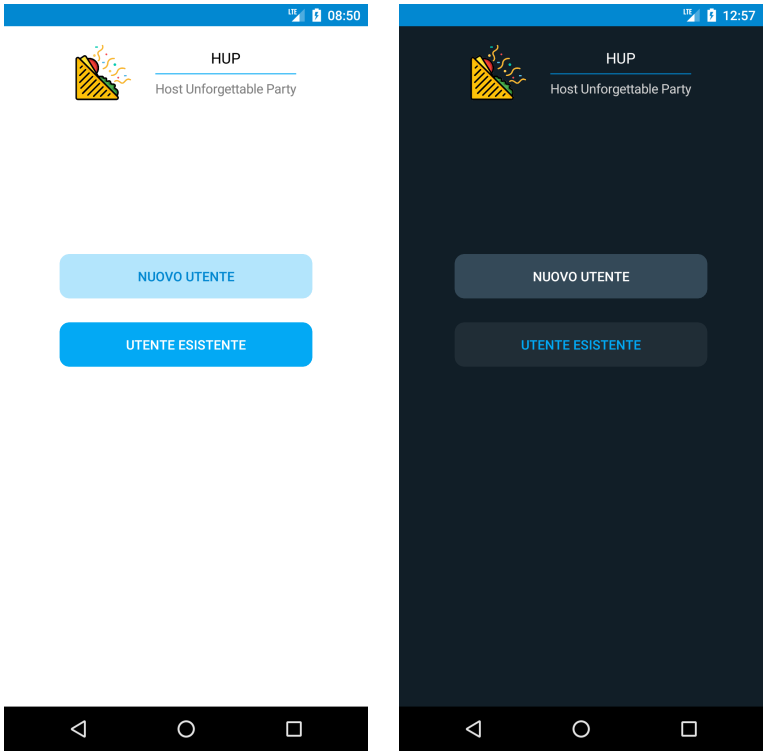


Figura 5.4: Prima scelta in LightMode e in DarkMode

Nella pagina seguente (Figura 5.5) troviamo tre pulsanti per le tre scelte possibili per registrarsi/accedere nella nostra applicazione, ovvero:

- Google;
- Facebook;
- E-mail.

In questa pagina si sceglie il tipo di account ma non si effettua subito la registrazione. Infatti, solo dopo aver completato tutti i passaggi per inserire i dati del profilo si procederà ad effettuare la registrazione effettiva. Mentre per quanto riguarda il login, una volta scelto l'account, si va direttamente all'inserimento dei dati già registrati (e-mail e password o account Google/Facebook).

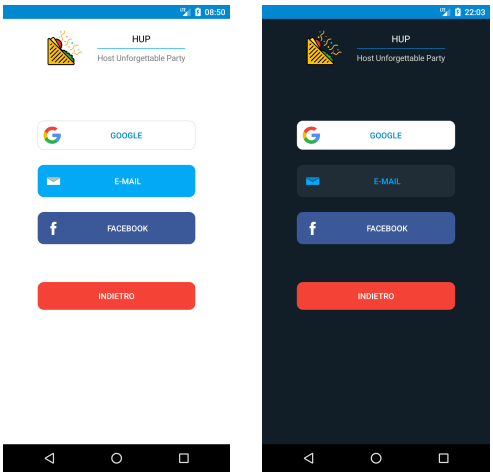


Figura 5.5: Scelta dell'account in LightMode e in DarkMode

La pagina `PhotoChoice` (Figura 5.6) fa parte del "percorso" di registrazione. Tramite essa si iniziano ad inserire i primi dati per la creazione del profilo. In particolare, in questa pagina, possiamo inserire il numero di telefono e la foto del profilo. Quest'ultima viene salvata solo alla fine della registrazione nello storage di Firebase. Con il pulsante "Indietro" si torna alla prima scelta "`FirtstChoice`" cancellando tutti i dati precedentemente inseriti; prima di procedere si apre un AlertDialog che ci chiede conferma dell'operazione prima di eseguirla.

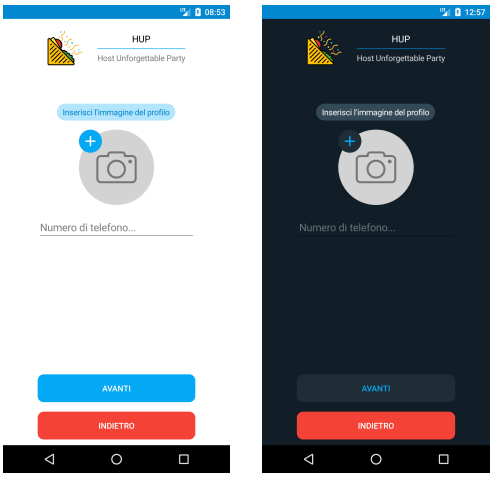


Figura 5.6: Scelta della foto in LightMode e in DarkMode

Nella pagina successiva (Figura 5.7) vi è la scelta della regione in cui si trova l'utente in quel momento; questa ci servirà, poi, per decretare se un utente può

organizzare un evento e chi può invitare in base alle normative sui colori delle regioni auttalmente in vigore.

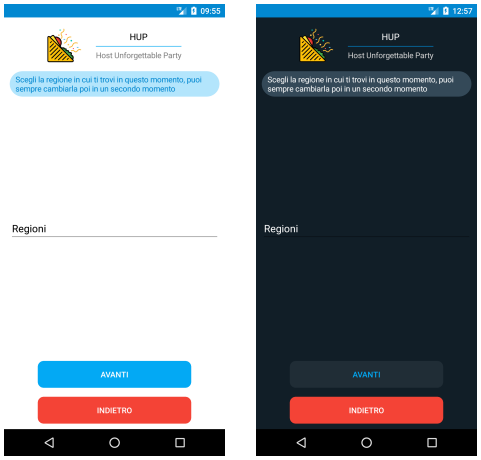


Figura 5.7: Scelta della regione in LightMode e in DarkMode

La prossima pagina (Figura 5.8) è una delle parti fondamentali della nostra app. Infatti, in essa vengono inserite le proprie intolleranze, scegliendole da una lista di 100 alimenti (Figura 5.9). Tali intolleranze verranno, poi, notificate agli organizzatori degli eventi in modo che questi ultimi possano tenerne conto nella creazione del menù. Queste intolleranze ovviamente possono essere modificate in un secondo momento. Con il pulsante "Indietro" si torna alla prima scelta "`FirtstChoice`" cancellando tutti i dati precedentemente inseriti; prima di procedere si apre un AlertDialog che ci chiede conferma dell'operazione prima di eseguirla.

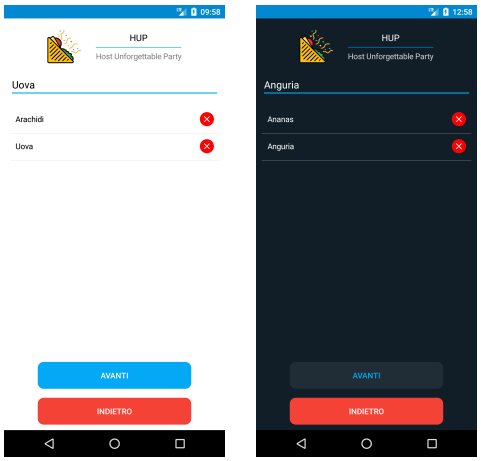


Figura 5.8: Inserimento delle intolleranze in LightMode e in DarkMode

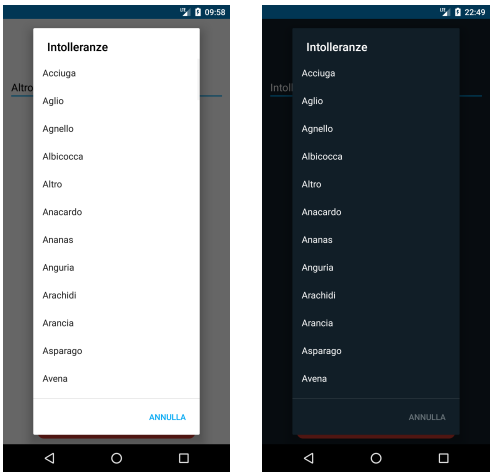


Figura 5.9: Lista delle intolleranze nell'app in LightMode e in DarkMode

Se all'inizio della registrazione è stato cliccato il pulsante e-mail, alla fine dell'inserimento dei dati si arriverà a questa pagina dove si potranno inserire:

- nome e cognome;
- e-mail;
- password.

Come possiamo vedere abbiamo 2 configurazioni di questa pagina che sono, rispettivamente, una per la registrazione (Figura 5.10) e una per il login (Figura 5.11). Con il pulsante "Indietro" si torna alla prima scelta "`FirtstChoice`" cancellando tutti i dati precedentemente inseriti. Prima di procedere si apre un AlertDialog che ci chiede conferma dell'operazione prima di eseguirla.

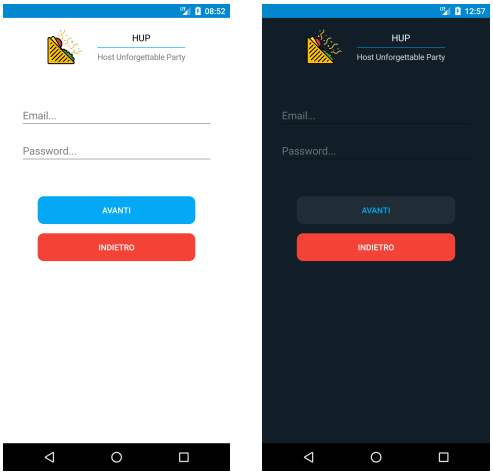


Figura 5.10: Schermata della email per il login in LightMode e in DarkMode

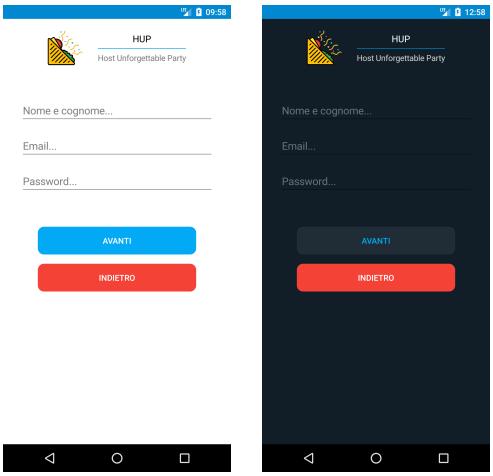


Figura 5.11: Schermata email per il signin in LightMode e in DarkMode

Poichè per caricare nel database Firestore tutte le informazioni inserite nella registrazione potrebbe essere necessario qualche secondo (in base alla velocità della rete e alla potenza del telefono), è stata inserita una schermata di caricamento (Figura 5.12).

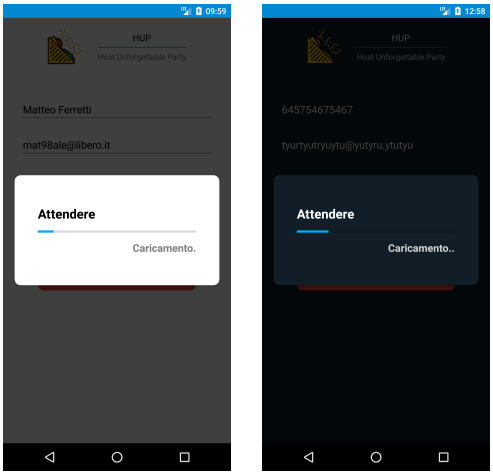


Figura 5.12: Schermata di caricamento in LightMode e in DarkMode

5.2 Home

Una volta che ci si è registrati o una volta che è stato effettuato il login, si viene rediretti alla `MainPage`. Questa pagina è costituita da tre Tab. La prima consente

di vedere la lista degli eventi a cui l'utente è stato invitato oppure di cui è l'organizzatore (Eventi Attivi, Figura 5.13). La seconda consente di vedere gli eventi che l'utente ha rifiutato o che si sono già svolti (Storia, Figura 5.14). Infine, l'ultima visualizza i dati personali (Profilo, Figura 5.15) dando all'utente anche la possibilità di cambiarli.

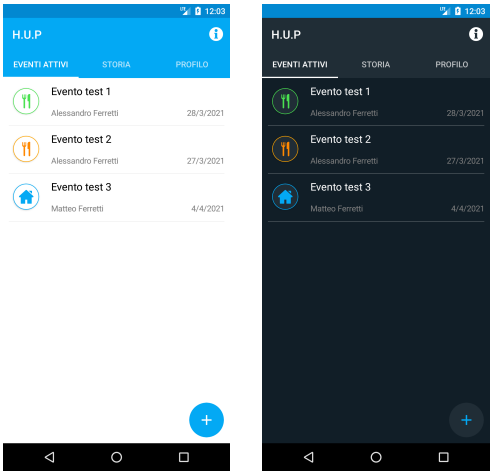


Figura 5.13: Eventi attivi in LightMode e in DarkMode

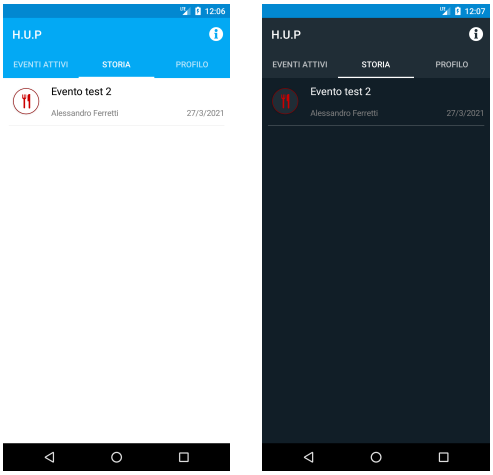


Figura 5.14: Storia degli eventi in LightMode e in DarkMode

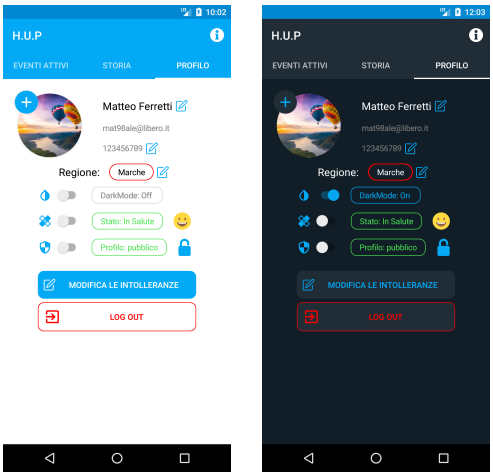


Figura 5.15: Profilo in LightMode e in DarkMode

Nella Tab degli Eventi Attivi, oltre alla lista degli eventi, è presente, in basso a destra, il pulsante tramite il quale si viene rimandati nella pagina di creazione dell'evento. In alto a destra, vi è il pulsante che rimanda alla pagina delle informazioni riguardanti il Covid-19. In questa pagina vi sono 4 pulsanti (Figura 5.16):

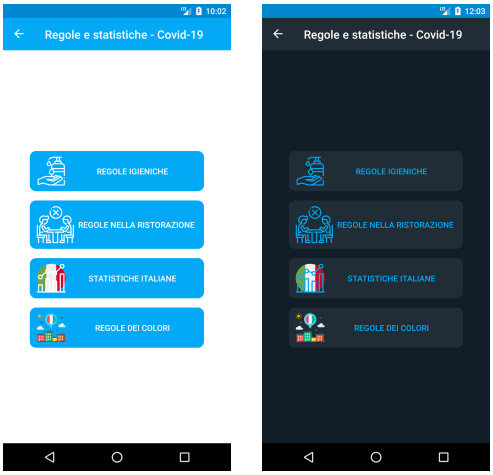


Figura 5.16: Pagina delle informazioni in LightMode e in DarkMode

- *Regole igieniche*: premendo questo pulsante vengono visualizzate le regole igieniche stilate dal governo (Figura 5.17);

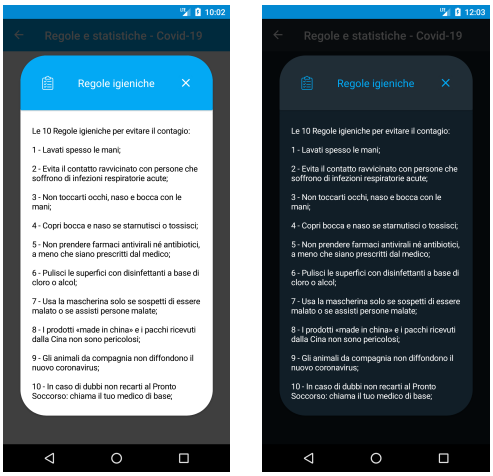


Figura 5.17: Regole igieniche in LightMode e in DarkMode

- *Regole nella ristorazione*: premendo questo pulsante vengono visualizzate le regole della ristorazione stilate dal governo (Figura 5.18);

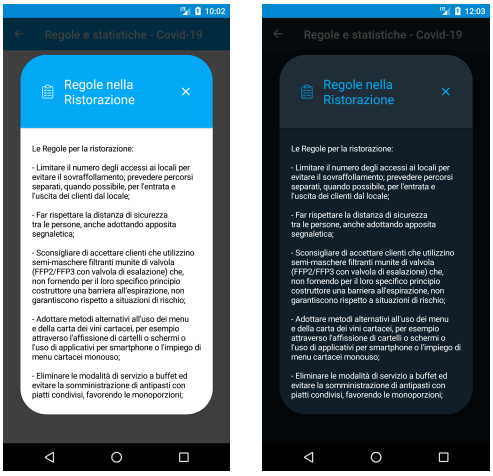


Figura 5.18: Regole nella ristorazione in LightMode e in DarkMode

- *Statistiche italiane*: premendo questo pulsante si apre il sito web della Protezione Civile italiana dove sono riportate le statistiche sulla situazione in Italia (Figura 5.19);
- *Regole dei colori*: premendo questo pulsante si apre il sito web del governo dove sono riportate tutte le normative riguardanti i colori delle regioni (Figura 5.19).

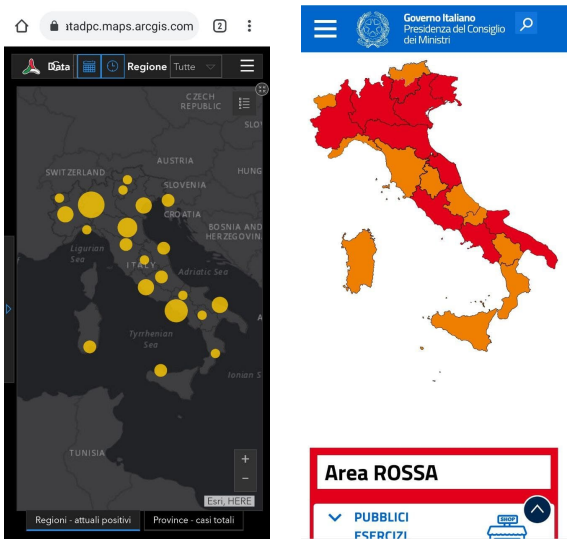


Figura 5.19: Pagine web relative alle statistiche e ai colori delle regioni

Inoltre, facendo swipe verso il basso, si può aggiornare la lista stessa facendo apparire, quindi, la rotellina di caricamento al termine della quale, se ci sono nuovi eventi, verranno aggiunti.

Questa sezione è costituita dalla propria immagine del profilo che si può modificare premendo sull'immagine stessa. Sotto di essa compaiono il proprio nome, il numero di telefono, la regione attuale dell'utente (anch'essi modificabili) e l'email. Sotto le informazioni principali vi sono 3 switch che servono per impostare alcuni parametri del proprio profilo. Tali parametri sono i seguenti:

- *DarkMode*, che ci permette di scegliere il tipo di tema per la nostra applicazione, ovvero se si preferisce il tema scuro oppure quello standard.
- *Stato*, che indica il nostro stato di salute, in modo che, nel caso un utente non si senta bene, il corrispettivo stato venga posto ad ammalato (Figura 5.20). Ciò consentirà a chi organizza un evento di sapere che quella persona non va invitata perchè "malata";
- *Profilo*, con questo pulsante diamo la possibilità ad un utente di rendere privato il proprio profilo (Figura 5.21). In questo modo non apparirà più nella lista dei possibili ospiti, ma, per invitarlo, bisognerà inserire la sua mail o il suo numero di telefono per intero.

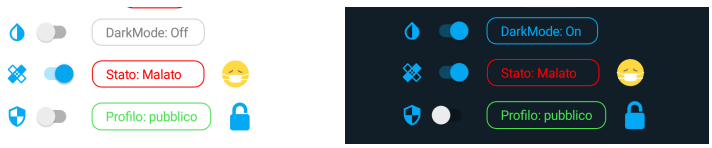


Figura 5.20: Stato di salute in LightMode e in DarkMode

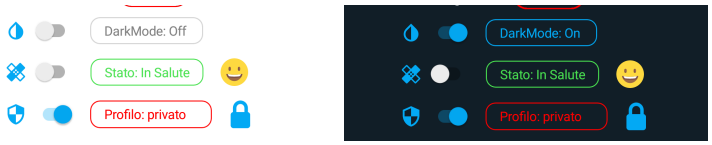


Figura 5.21: Stato profilo in LightMode e in DarkMode

Infine, sono presenti 2 pulsanti:

- *modifica intolleranze*, in cui si possono modificare le proprie intolleranze dando, quindi, la possibilità all'utente che non le aveva inserite di poterle inserire in un secondo momento;
- *log out*, tramite il quale si può chiudere la propria sessione di login ed essere rimandati alla pagina iniziale che permette di scegliere tra l'accesso o la registrazione.

La visualizzazione delle icone relative ad ogni evento è stata organizzata in questo modo:

- Se l'utente è l'organizzatore allora vedrà come immagine dell'evento l'icona della casa con colore Blu.
- Se è un invitato vedrà l'icona della forchetta con il colore degli sfondi differenti a seconda della propria decisione a partecipare all'evento:
  - se l'immagine è arancione vuol dire che la decisione è ancora da prendere;
  - se l'immagine è verde vuol dire che l'invito è stato accettato;
  - se l'immagine è rossa vuol dire che l'invito è stato rifiutato (in questo caso si trova esclusivamente nella tab "Storia").

Ogni evento è cliccabile e, una volta premuto su di esso, si viene rimandati alla pagina che ne mostra i dettagli.

Questa pagina è costituita da 3 tab in cui si possono vedere le informazioni relative all'evento selezionato. Ogni tab contiene informazioni di tipo diverso; più specificatamente:

- Il tab *Menu* contiene il menu che verrà servito il giorno dell'evento (Figura 5.22).
- Il tab *Posizione e Contatti* contiene i dati relativi all'utente che ha organizzato l'evento (immagine del profilo, nome, e-mail e numero di telefono), nonché la data, l'ora e la posizione dell'evento (Figura 5.23).
- Il tab che si trova al centro è quello che contiene la lista degli invitati (Figura 5.24); per ciascun invitato vengono visualizzati l'immagine del profilo, il nome, l'e-mail, il numero di telefono e, inoltre, lo stato dell'invito rappresentato con dei cerchi di colori differenti. Con il colore arancione viene indicato che la decisione non è stata ancora presa, con il colore verde viene indicato che l'evento è stato accettato mentre con il rosso che l'utente non parteciperà all'evento.

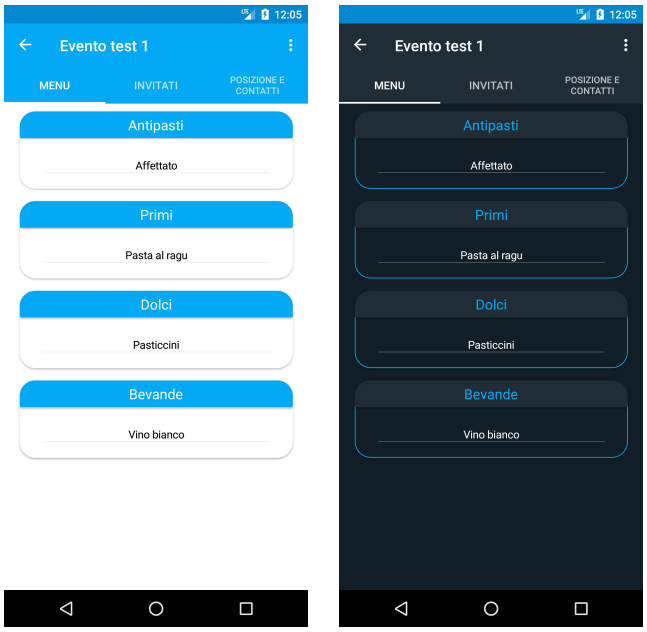


Figura 5.22: Menù evento in LightMode e in DarkMode

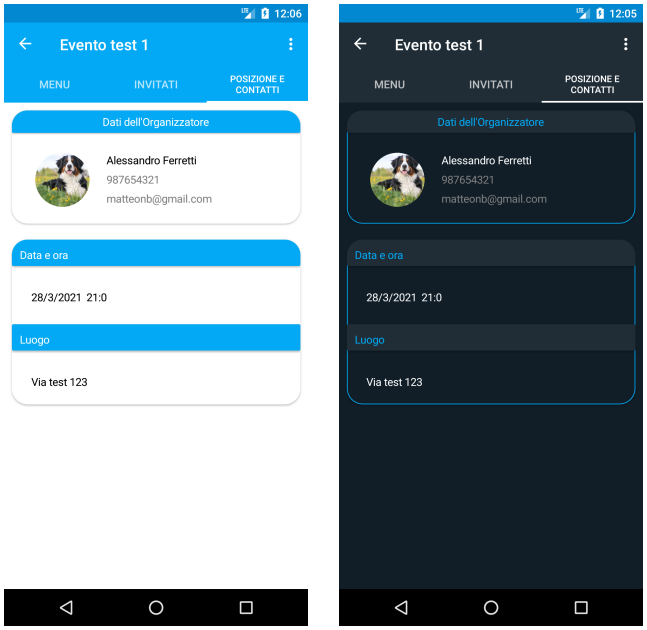


Figura 5.23: Posizione e contatti Evento in LightMode e in DarkMode

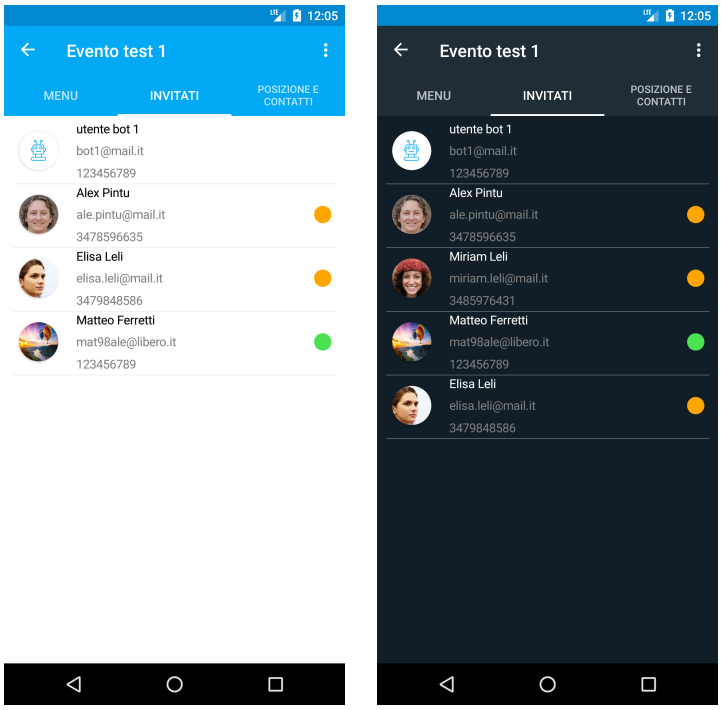


Figura 5.24: Lista invitati evento in LightMode e in DarkMode

La visualizzazione e le funzionalità di questa pagina cambiano a seconda del ruolo dell'utente all'interno dell'evento organizzato. Per questo motivo verrà mostrata, di seguito, la pagina distinguendo i vari casi e analizzandoli nel dettaglio:

- *Utente Organizzatore*: se il ruolo dell'utente è quello di "organizzatore" verrà visualizzato il menù con le voci *Modifica Evento* ed *Elimina Evento* (Figura 5.25). Se si sceglie di modificare l'evento, si aprirà la stessa pagina per la creazione dell'evento (Figura 5.26), ma con i dati già inseriti, in modo da poterli modificare. Alla pressione del pulsante per l'eliminazione, invece, viene aperto un AlertDialog in cui si può confermare o annullare l'eliminazione. In caso affermativo, l'evento non sarà più accessibile e verrà eliminato definitivamente dal database.

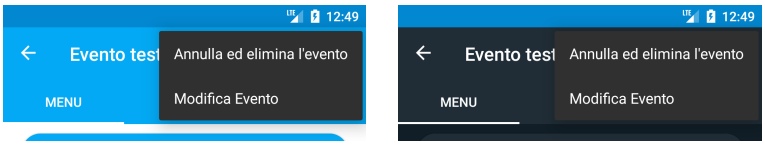


Figura 5.25: Menù organizzatore in LightMode e in DarkMode

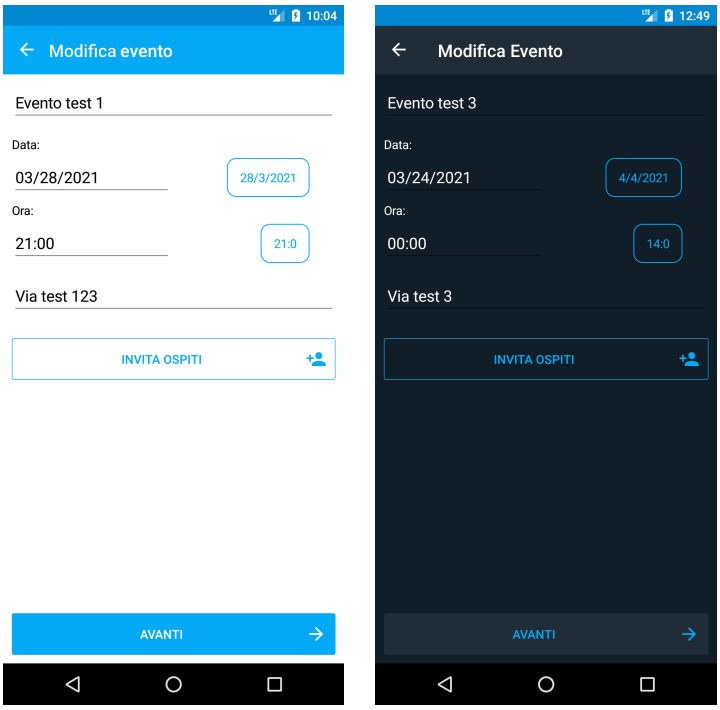


Figura 5.26: Modifica evento in LightMode e in DarkMode

- *Utente Invitato*: se il ruolo dell'utente è quello di "invitato" allora abbiamo più azioni e menù diversi:
  - *Decisione ancora non presa*: la prima cosa che è possibile fare è quella di accettare o rifiutare l'invito all'evento. Questa possibilità è data da 2 pulsanti presenti nella NavigationBar, uno di colore verde, che permette all'utente di accettarlo, e l'altro, di colore rosso che gli permette di rifiutarlo (Figura 5.27).

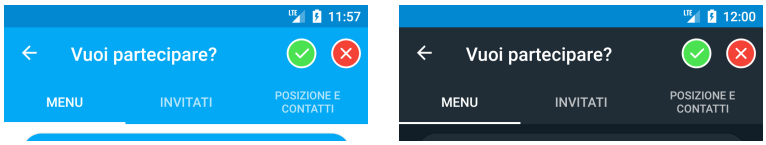


Figura 5.27: Scelta partecipazione ad un evento in LightMode e in DarkMode

Se l'utente accetta l'invito, prima di poter partecipare all'evento, gli viene sottoposta un'autocertificazione (Figura 5.28) da compilare. Questa certifica lo stato di salute dell'invitato; solo una volta verificata la sua idoneità, egli potrà partecipare all'evento.

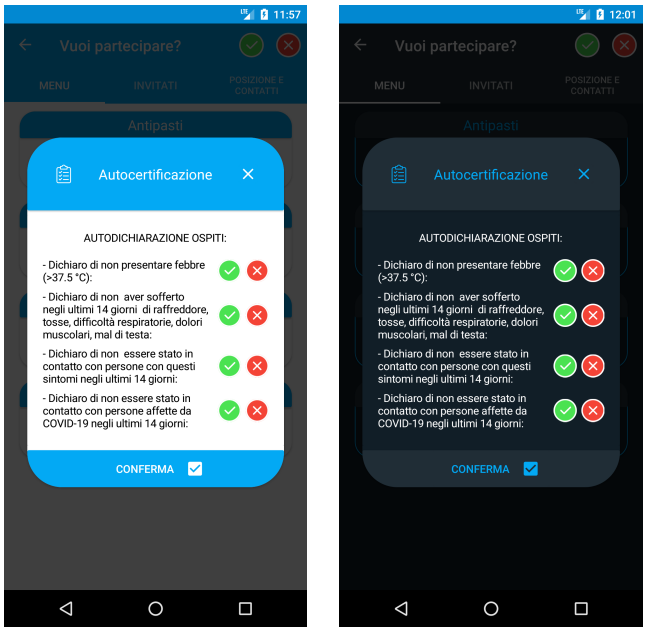


Figura 5.28: Autocertificazione in LightMode e in DarkMode

– *Invito accettato*: nel caso in cui l'utente accetti l'invito all'evento, verrà visualizzato un menù con l'opzione "Abbandona" (Figura 5.29) che dà la possibilità all'utente di lasciarlo. È stata realizzata questa opzione per permettere all'utente di abbandonare l'evento nell'ipotesi di un cambio di programma dovuto a eventi inaspettati. Alla pressione di questa voce, viene visualizzato, anche in questo caso, un AlertDialog per confermare o meno la propria volontà. In caso di abbandono, l'evento verrà visualizzato nella lista degli eventi della tab "Storia" con un'icona con immagine rossa.

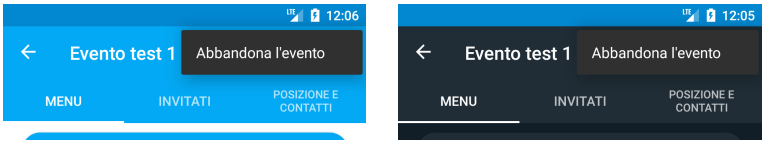


Figura 5.29: Menù invito accettato in LightMode e in DarkMode

– *Invito rifiutato*: nel caso in cui l'utente rifiuti l'invito all'evento, quest'ultimo verrà visualizzato nella sezione "Storia"; da qui, alla pressione dell'evento, si potranno ancora visualizzare i dati che lo caratterizzano. In aggiunta a ciò sarà presente un menù contenente al proprio interno la voce "elimina dalla lista" (Figura 5.31). Premendo su quest'ultima si darà la possibilità all'utente di eliminare l'evento dalla lista contenuta nella sezione "Storia" evitando, quindi, un eventuale accumulo di eventi rifiutati.

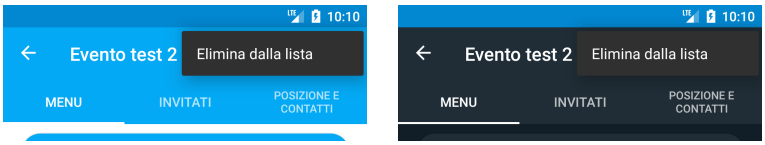


Figura 5.30: Menù invito non accettato in LightMode e in DarkMode

5.3 Creazione di un nuovo evento

La pagina NewEvent presenta 4 label per l'inserimento dei campi Nome, Data, Ora ed Indirizzo dell'evento (Figura 5.31).

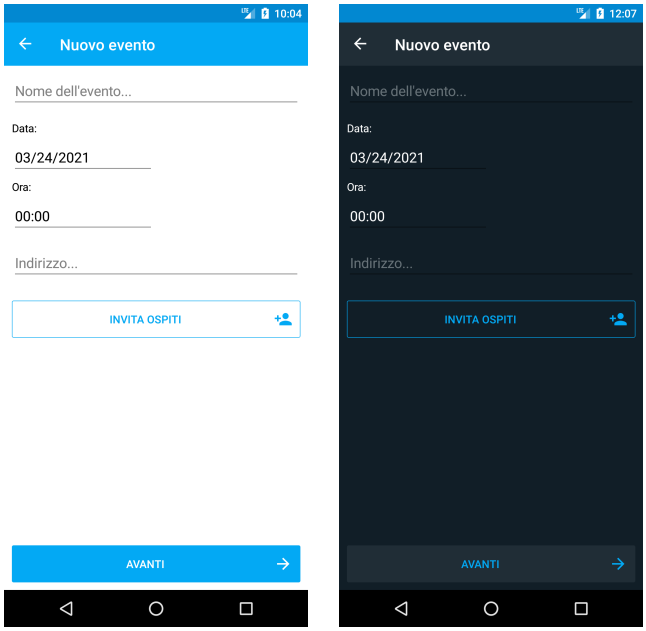


Figura 5.31: Pagina nuovo evento in LightMode e in DarkMode

In particolare, per quanto riguarda i campi Data ed Ora, al fine di evitare che l'utente possa digitarli in maniera errata, sono state disabilitate le label relative, in modo che al click aprono, rispettivamente, un DatePicker (Figura 5.32) ed un TimePicker (Figura 5.33). A questo punto, una volta selezionate data e ora, queste vengono riportate di lato in evidenza (Figura 5.34).

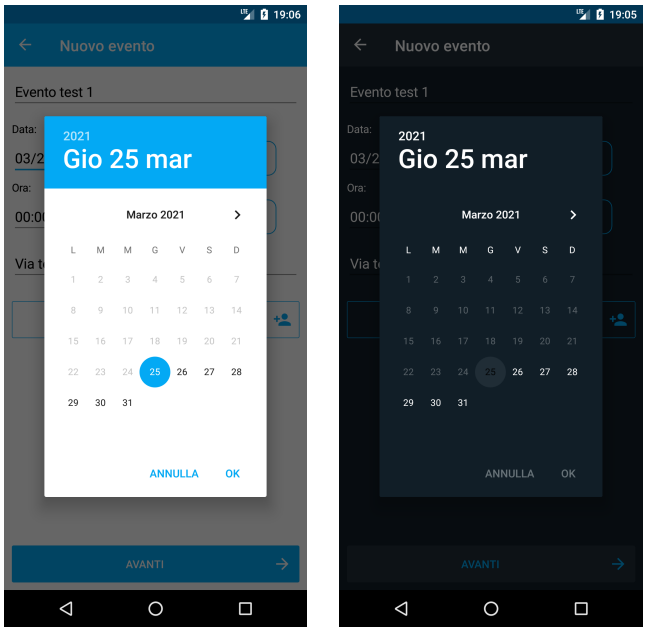


Figura 5.32: Data Picker in LightMode e in DarkMode

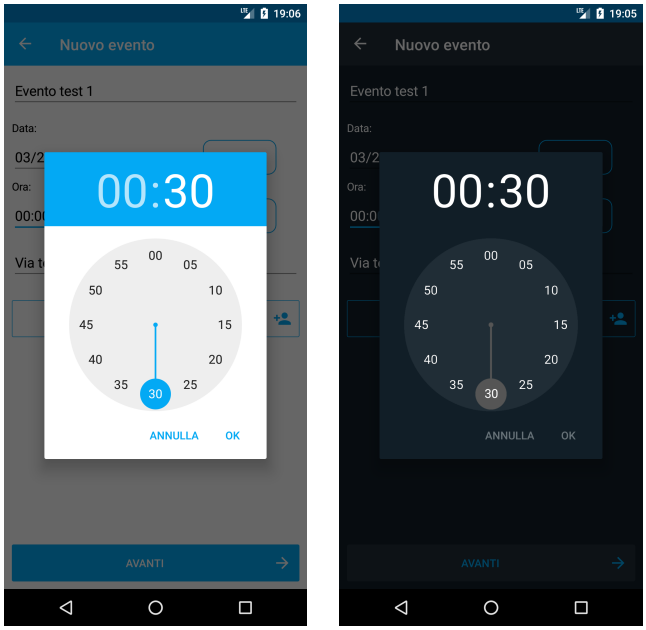


Figura 5.33: Time Picker in LightMode e in DarkMode

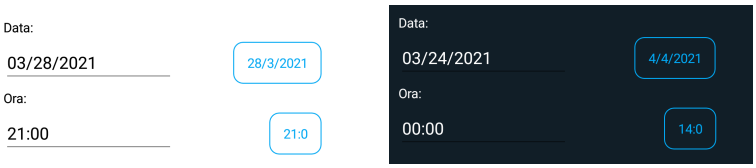


Figura 5.34: Data e ora in LightMode e in DarkMode

Il pulsante *INVITA OSPITI* permette di passare alla pagina `ContactEntry` mentre il pulsante *AVANTI* permette il passaggio alla pagina `FoodIntolerance`, non prima che i 4 campi siano riempiti e che la lista degli invitati contenga almeno un elemento.

La pagina `ContactEntry` è costituita principalmente da una ListView dove ogni riga corrisponde ad un invitato all'evento. Chiaramente, al primo accesso, la lista sarà vuota (Figura 5.35). Dopo l'inserimento degli invitati, ogni riga sarà costituita da 3 label relative a nome, e-mail e cellulare, un'image contenente l'immagine del profilo (se esistente) ed, infine, un pulsante per l'eliminazione del contatto dalla lista (Figura 5.36). È presente, inoltre, una SearchBar che ci permette di filtrare gli elementi della lista degli invitati per nome, email oppure numero di cellulare.

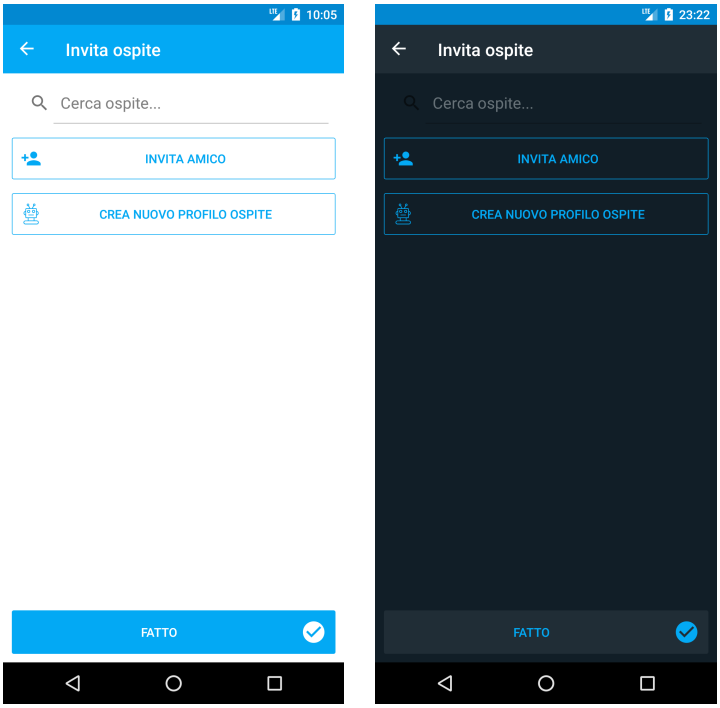


Figura 5.35: Lista degli invitati vuota in LightMode e in DarkMode

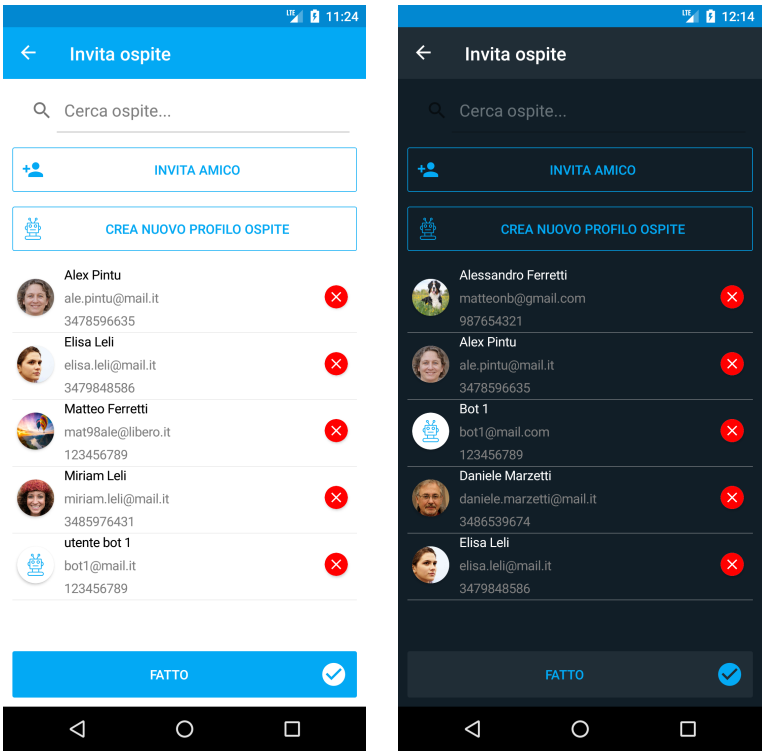


Figura 5.36: Lista invitati piena in LightMode e in DarkMode

I pulsanti *INVITA AMICO* e *CREA NUOVO PROFILO OSPITE* mandano, rispettivamente, alle pagine `ExistingContact` e `NewContact`, mentre, invece, il pulsante *FATTO*, posto in fondo alla pagina, consente di ritornare a `NewEvent`, passando ad esso la lista degli invitati.

La pagina `ExistingContact` (Figura 5.37) è costituita anch'essa da una ListView dove ogni riga rappresenta uno degli account registrati all'interno dell'applicazione. Le righe, oltre a contenere le tre label relative a nome, e-mail e cellulare e alla image per l'eventuale immagine del profilo, presentano una CheckBox per la selezione del contatto da aggiungere alla lista invitati. Attenzione però agli utenti che hanno come immagine del profilo un'emoticon con la mascherina perchè questo significa che il suo stato di salute è malato e quindi l'applicazione impedirà di invitarlo (Figura 5.38). Anche in questa pagina è presente una SearchBar che ci permette di filtrare gli elementi della lista degli invitati per nome, e-mail oppure numero di cellulare.

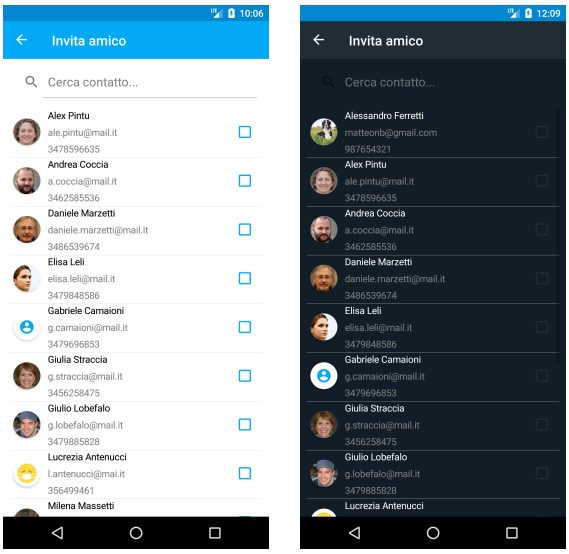


Figura 5.37: Lista utenti in LightMode e in DarkMode

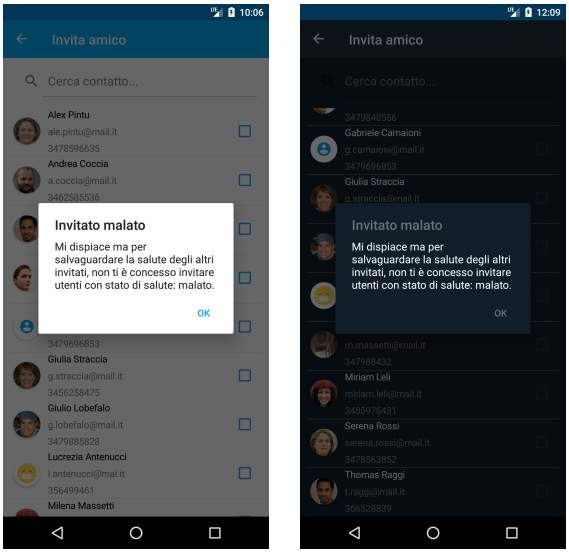


Figura 5.38: Alert invitato malato in LightMode e in DarkMode

Alla creazione della pagina è necessario il download dei contatti dal database Firestore. Poiché, in presenza di molteplici contatti, l'operazione potrebbe richiedere un allungamento dei temi di attesa, per segnalare l'effettivo svolgimento del download è stata inserita una ProgressBar che scompare al termine dell'operazione (Figura 5.39).

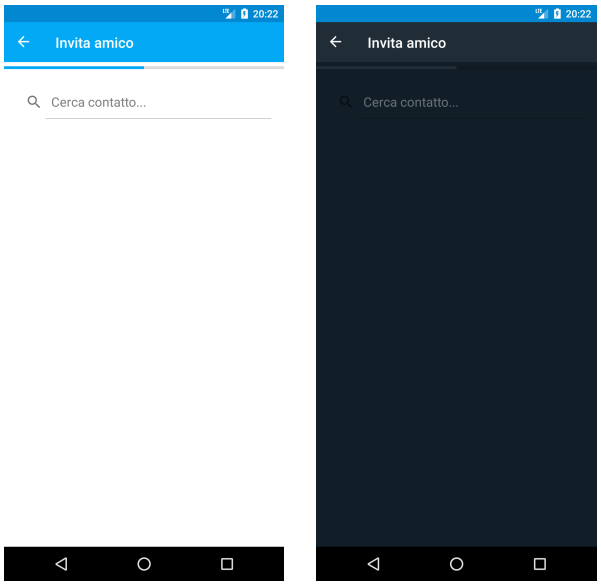


Figura 5.39: ProgressBar in LightMode e in DarkMode

La spunta di una CheckBox rende visibile sul fondo dell'activity il pulsante *AGGIUNGI INVITATI*, che consente l'aggiunta dei contatti selezionati alla lista degli invitati (Figura 5.40).

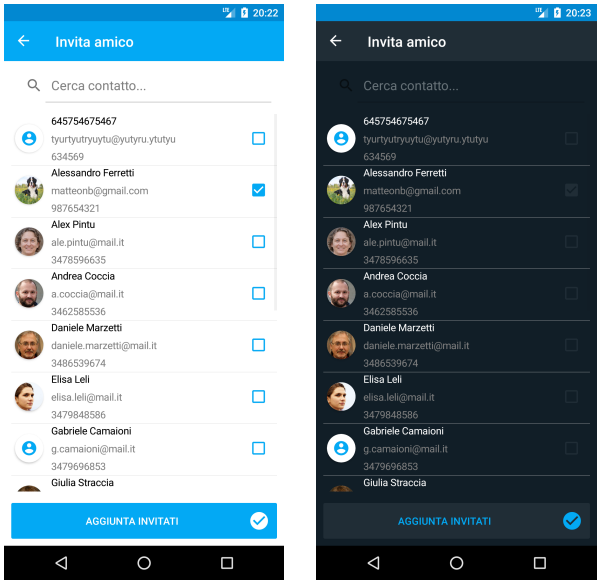


Figura 5.40: Pulsante aggiungi invitati in LightMode e in DarkMode

La pagina `NewContact` (Figura 5.41), invece, consente di aggiungere alla lista degli invitati persone che non hanno scaricato la nostra applicazione e quindi non possiedono un account. L'interfaccia utente dispone di 3 label, per l'inserimento dei tre campi obbligatori relativi a nome, e-mail e cellulare, ed un picker, per selezionare da una lista salvata sul database Firestore le intolleranze alimentari segnalate dagli invitati che andranno ad aggiungersi ad una ListView sottostante.

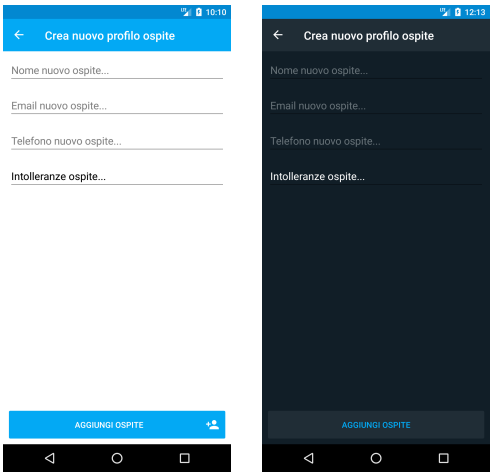


Figura 5.41: Creazione nuovo invitato in LightMode e in DarkMode

Prima di passare alla lista delle intolleranze, però, compare un AlertDialog che ci chiede se vogliamo scegliere la disposizione degli invitati (Figura 5.42).

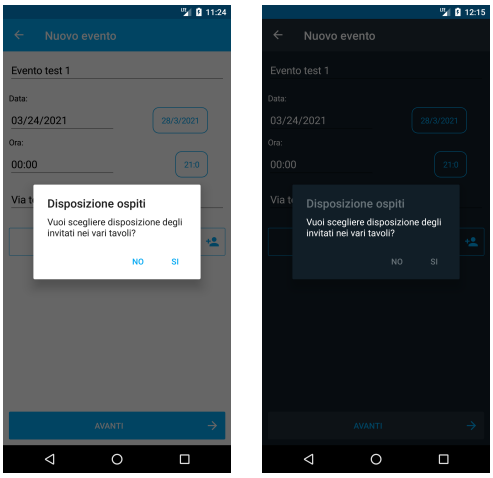


Figura 5.42: Alert per il posizionamento degli invitati in LightMode e in DarkMode

Se si da risposta affermativa appare una pagina dove vi sono soltanto tre pulsanti nella Appbar superiore (Figura 5.43):

- *Aggiungi tavolo*: tramite questo pulsante viene aggiunto una tab tavolo alla lista dei tavoli inserendo la sua larghezza e la sua lunghezza (Figura 5.44);
- *Rimuovi tavolo*: tramite questo pulsante è possibile rimuovere il tavolo corrispondente al tab attualmente selezionato;
- *Fatto*: tramite questo pulsante vengono salvate le posizioni degli invitati e si passa alla pagina successiva;

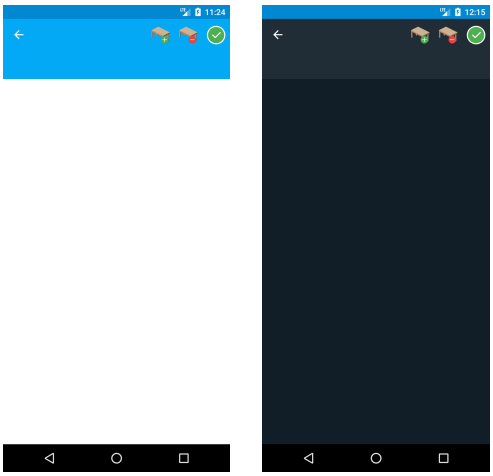


Figura 5.43: Pagina per il posizionamento degli invitati in LightMode e in DarkMode

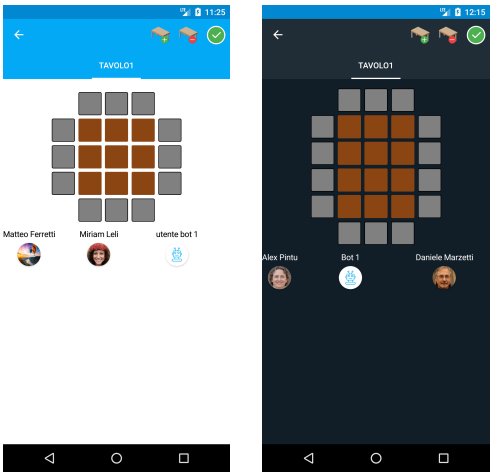


Figura 5.44: Tavolo di esempio in LightMode e in DarkMode

Attenzione, però, quando posizioniamo gli invitati perchè non si possono mettere 2 invitati vicini a meno che non appartengano ad uno stesso nucleo familiare (Figura 5.45), e non si possono ovviamente posizionare gli invitati sopra al tavolo (Figura 5.46).

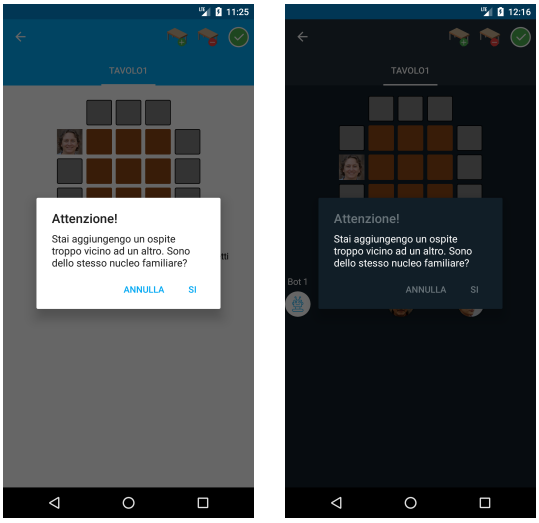


Figura 5.45: Alert per il posizionamento degli invitati dello stesso nucleo familiare in LightMode e in DarkMode

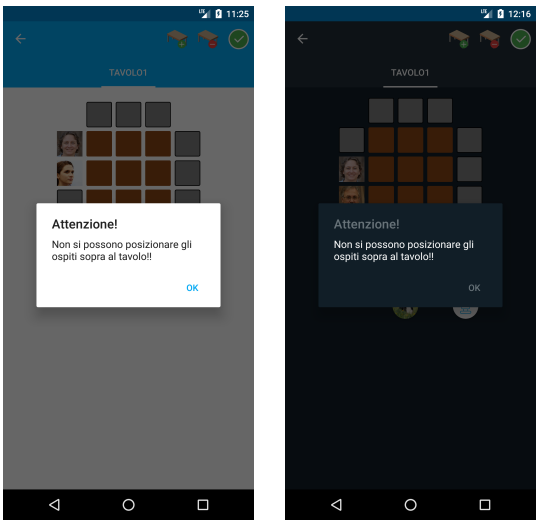


Figura 5.46: Alert per il posizionamento degli invitati sopra al tavolo in LightMode e in DarkMode

La pagina `FoodIntolerance` mostra tutte le intolleranze alimentari dichiarate dai membri della lista invitati, specificandone anche il numero, all'interno di una ListView. Al di sopra di essa, è presente una label il cui aspetto varia a seconda della presenza (Figura 5.47) o meno (Figura 5.48) di intolleranze alimentari fra gli ospiti.

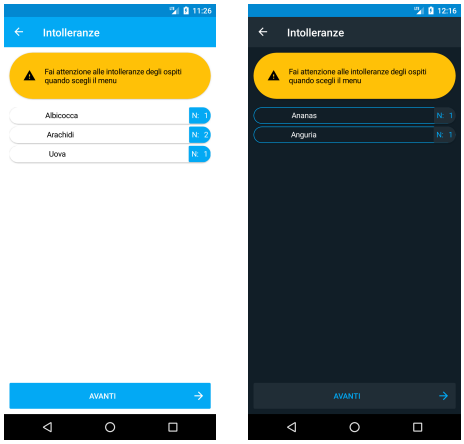


Figura 5.47: Lista delle intolleranze degli ospiti in LightMode e in DarkMode

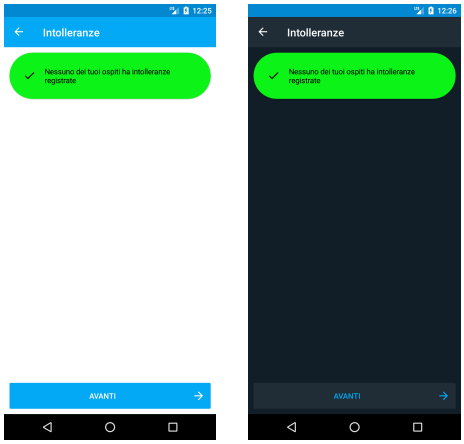


Figura 5.48: Lista delle intolleranze degli ospiti vuota in LightMode e in DarkMode

Una volta inserite tutte le informazioni riguardo l'evento, e una volta formata la lista degli invitati, il passo successivo e conclusivo è quello della creazione del menù (Figura 5.49). Questa pagina presenta un'eventuale label in presenza di intolleranze alimentari fra gli ospiti mostrate nuovamente da una finestra di Dialog che si apre al suo click (Figura 5.50).

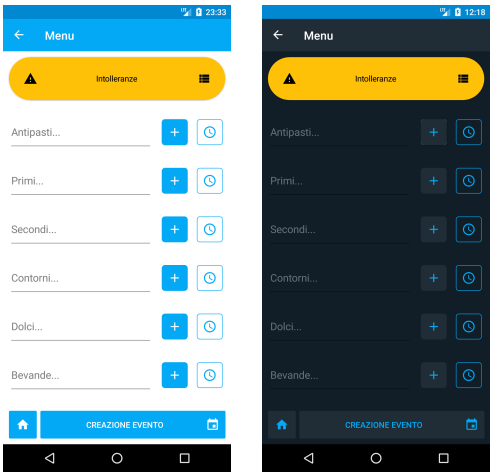


Figura 5.49: Creazione del menù in LightMode e in DarkMode

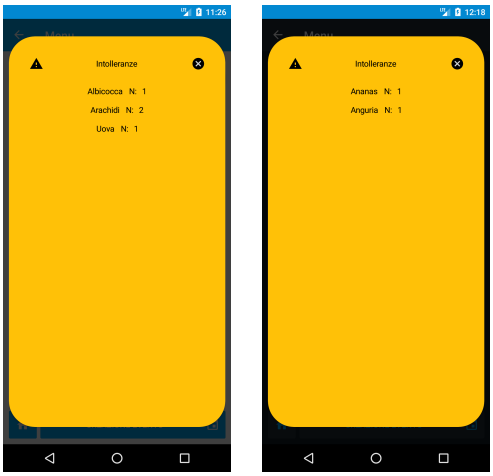


Figura 5.50: Dialog che racchiude tutte le intolleranze in LightMode e in DarkMode

La pagina è costituita da una label dove inserire la portata divisa per tipo (antipasti, primi, secondi, contorni, dolci, bevande); a fianco vi è anche un pulsante che fa comparire un picker che mostra tutte le portate di un determinato tipo già realizzate in eventi passati e già salvate nel database (Figura 5.51). In ogni caso, sia che si sia deciso di riproporre un piatto già presente nel database, o sia che si sia optato per inserirne uno completamente nuovo, sarà necessario premere il tasto "+" sulla destra al fine di salvare il nuovo piatto nella ListView sottostante.

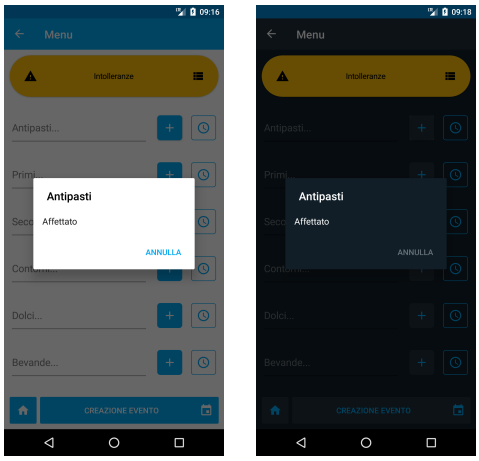


Figura 5.51: Portate realizzate precendentemente in LightMode e in DarkMode

Terminata la creazione del menù, se si clicca il pulsante *HOME* si tornerà alla `MainPage`; se, invece, si preme il pulsante *CREATE EVENT* potrà aprirsi una eventuale finestra di dialogo per avvertirci se un piatto scelto è stato già proposto in un precedente evento allo stesso invitato, con indicazione della data relativa (Figura 5.52).

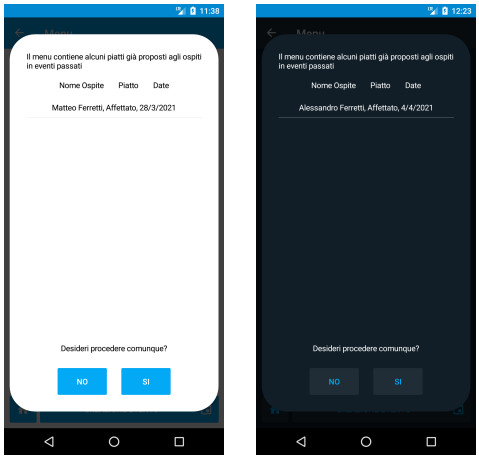


Figura 5.52: Lista riassunto dei piatti preparati più volte ad uno stesso invitato in LightMode e in DarkMode

Una volta deciso di continuare nonostante l'avvertimento, l'evento verrà salvato nel database, unitamente ai nuovi piatti creati per l'occasione che, quindi, verranno inseriti tra i suggerimenti del picker per i prossimi eventi. Terminata la creazione dell'evento, si torna automaticamente alla `MainPage`.

6

Discussione in merito al lavoro svolto

Questo capitolo si compone di tre sezioni: nelle prime due si presenterà la SWOT analysis che ci consentirà di effettuare una valutazione del sistema realizzato. Nell'ultima sezione esploreremo il panorama odierno delle soluzioni correlate cercando di individuare le possibili rivali della nostra applicazione.

6.1 SWOT Analysis

SWOT (Figura 6.1) è un acronimo che sta per Strenghts (punti di forza), Weaknesses (debolezze), Opportunities (opportunità) e Threats (minacce).

Figura 6.1: Significato dell'acronimo SWOT

L'analisi SWOT (SWOT Analysis), detta anche Matrice SWOT (Figura 6.2), analizza un progetto o un business focalizzandosi su determinati fattori, aiutando a mettere a fuoco le caratteristiche distintive della propria attività e del mercato di riferimento [12].

Figura 6.2: Matrice SWOT

Si noti come i punti di forza e le debolezze fanno riferimento all'ambiente interno del sistema, mentre le opportunità e le minacce a quello esterno. Questi diagrammi sono particolarmente utili quando è necessario decidere se avviare o meno un'impresa. Aiutano a visualizzare chiaramente pro e contro, evidenziando tutti gli aspetti positivi e negativi di un progetto. Con le informazioni raccolte da una SWOT Analysis è più facile capire se procedere e come muoversi.

Applicata al progetto, l'analisi SWOT si compone di quattro passi:

- Nell'area "punti di forza" (in alto a sinistra) andranno inseriti tutti i tratti positivi che caratterizzano il nostro business distinguendolo dai competitor. In cosa eccelle la nostra azienda? Cosa la rende unica? Quali sono le nostre migliori qualità?
- I punti deboli (in alto a destra), come è facile immaginare, sono gli aspetti in cui si è più carenti. Quali sono le nostre aree di miglioramento? In cosa sono migliori i nostri concorrenti?
- Le opportunità, o rischi positivi (in basso a sinistra), sono le possibilità offerte dal mercato potenzialmente vantaggiose per il nostro business. È facile intuire che variano in base al mercato di riferimento e agli obiettivi aziendali, ma è importante saperle cogliere per capire quali azioni mettere in campo per sfruttarle.
- Le minacce, o rischi negativi (in basso a destra), infine, comprendono tutti gli eventuali ostacoli che potrebbero impedire la realizzazione della nostra idea di business. Il lancio di un nuovo prodotto o servizio concorrente, le regolamen-

tazioni del mercato, e così via; in altre parole, viene presa in considerazione qualsiasi cosa possa intralciare i nostri piani.

Una volta stilato un quadro completo della situazione, si esamina il diagramma e si analizzano i risultati. Questo passaggio spesso necessita del supporto di un consulente esterno che sappia giudarci nell'interpretazione corretta della matrice. Mettendo sulla bilancia aspetti positivi e negativi, se i primi superano i secondi si può proseguire tranquillamente con la propria idea e cominciare a metterla in pratica. Se, invece, gli aspetti negativi hanno un peso maggiore, probabilmente è il caso di rivedere il progetto o, nella peggiore delle ipotesi, abbandonarlo.

6.2 SWOT Analysis relativa all'app sviluppata

6.2.1 Strengths (Punti di forza)

I punti di forza individuati per il progetto sono i seguenti:

- *Grafica moderna*: uno degli aspetti su cui ci si è concentrati di più è quello grafico; infatti, uno dei punti fondamentali per invogliare un utente a scaricare un'app è la grafica.
- *Facilità d'uso*: un altro aspetto fondamentale doveva essere la facilità con cui un utente utilizzava l'applicazione. Poichè lo scopo dell'applicazione è organizzare feste ma anche, soprattutto, pranzi e cene a casa, un utente medio potrebbe essere la mamma, o la nonna addirittura, che invitava i suoi parenti a pranzo; per farle utilizzare questa applicazione dovevamo renderla il più intuitiva possibile.
- *Leggerezza*: chi spesso non si ritrova con quella brutta notifica dello "spazio di memoria quasi esaurito", e nonostante cancelli diversi file e applicazioni non scompare? La nostra applicazione è stata realizzata pensando anche a tutti quelli che hanno pochissimo spazio in memoria; infatti, è molto leggera perchè non mantiene dati salvati nel telefono, ma tutte le immagini e le informazioni sugli eventi vengono scaricate ogni volta che si apre l'applicazione dal nostro server.
- *Anti-Covid-19*: la situazione attuale la conosciamo tutti e, purtroppo, non possiamo fare altro se non adattarci alle normative che vengono rilasciate di volta in volta dal governo; ma la nostra applicazione è al passo con i tempi, le rispetta e le ricorda constantemente.

6.2.2 Weaknesses (Debolezze)

I punti di debolezza riscontrati nel progetto sono i seguenti:

- *Problemi di connessione*: la nostra applicazione è leggera, ma, come rovescio della medaglia, ha la necessità costante di una connesione ad Internet; infatti senza di essa, non è possibile prelevare dati dal database.
- *Immagini troppo grandi*: essendo un'app ibrida e non nativa sono stati usati dei pacchetti NuGet esterni per l'implementazione del download e dell'upload delle immagini del profilo degli utenti; questo comporta dei rallentamenti all'interno dell'app quando si ha necessità di caricare più immagini contemporaneamente.

- *Spazio limitato*: purtroppo lo spazio messo a disposizione da Google all'interno del suo database Firebase non è infinito; purtroppo esso prevede un numero di utenti giornalieri e download di dati limitato che non ci permette di avere una utenza elevata con il contratto base, obbligandoci a comprare, in caso di necessità, slot di memoria extra.

6.2.3 Opportunities (Opportunità)

- *Attuale*: la nostra applicazione si integra perfettamente con il contesto generale attuale; poche applicazioni sono contestualizzate alla situazione globale del Covid-19. Noi, però, ci tenevamo ad avere un'applicazione che potesse fare ciò che avevamo in mente rispettando le norme che vengono via via emanate.
- *Aumento della velocità*: grazie all'avanzamento tecnologico sarà possibile, in futuro, avere una connessione ad Internet veloce e sempre disponibile, che renderà più veloce e usufruibile la nostra applicazione.
- *Fine Covid-19*: quando la pandemia da Covid-19 sarà terminata e si tornerà pian piano alla normalità, le persone inizieranno ad uscire e ritrovarsi sempre con maggior frequenza, e quale modo migliore per organizzarsi se non utilizzando la nostra applicazione che potrebbe avere un boom grazie alla ripresa globale?

6.2.4 Threats (Minacce)

- *Fine Covid-19*: quando il Covid-19 scomparirà, però, saranno superflue molte delle sezioni integrate per prevenire i contagi, molte pagine diventeranno obsolete e ci sarà bisogno di un aggiornamento massiccio per eliminare tutte le sezioni che non serviranno più.
- *Assenza di interazioni*: purtroppo è inutile negarlo, al giorno d'oggi vi è sempre più interazione via Internet tramite social network, e un'applicazione che non permette di scambiare messaggi con gli utenti di uno stesso evento perde del fascino nei confronti di altre dove c'è maggiore interazione tra gli utenti, con scambio di messaggi, emoticon e foto varie.
- *Concorrenza sul mercato*: quale è la fascia di mercato dove si va a piazzare la nostra applicazione? Ci sono altre app simili che potrebbero farci concorrenza? A queste domande risponderemo nella prossima sezione.

6.3 Panoramica applicazioni concorrenti

Consultando il Play Store di Google possiamo trovare delle app simili alla nostra, come, per esempio:

- *My Party App*: questa applicazione (Figura 6.3) permette di creare eventi consultando una lista di luoghi, intrattenitori, produttori di torte, etc. per organizzare al meglio la propria festa. Vi è, anche, la possibilità di effettuare pagamenti tramite quest'app per raccogliere i soldi per un ipotetico regalo di compleanno (Figura 6.4).

Figura 6.3: My Party App

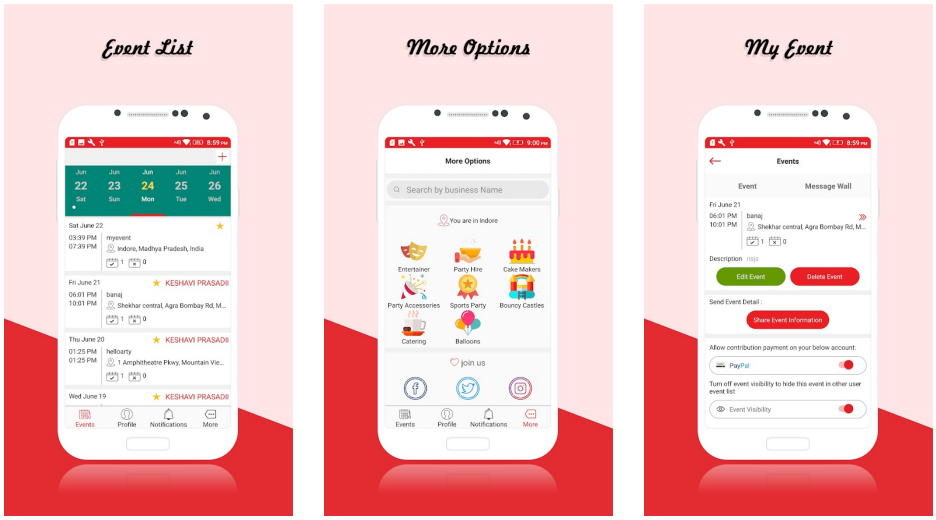


Figura 6.4: Alcuni screenshot dell'applicazione My Party App

- *Situa - Eventi e feste in casa*: questa applicazione (Figura 6.5) permette all'utente di creare dei party a casa propria come se fossero delle vere e proprie serate da discoteca, con un numero di posti limitato e un tema per la serata. Utilizzando la tua posizione sarà possibile individuare le feste che fanno al caso nostro più vicine a noi (Figura 6.6).

Figura 6.5: Situa - Eventi e feste in casa

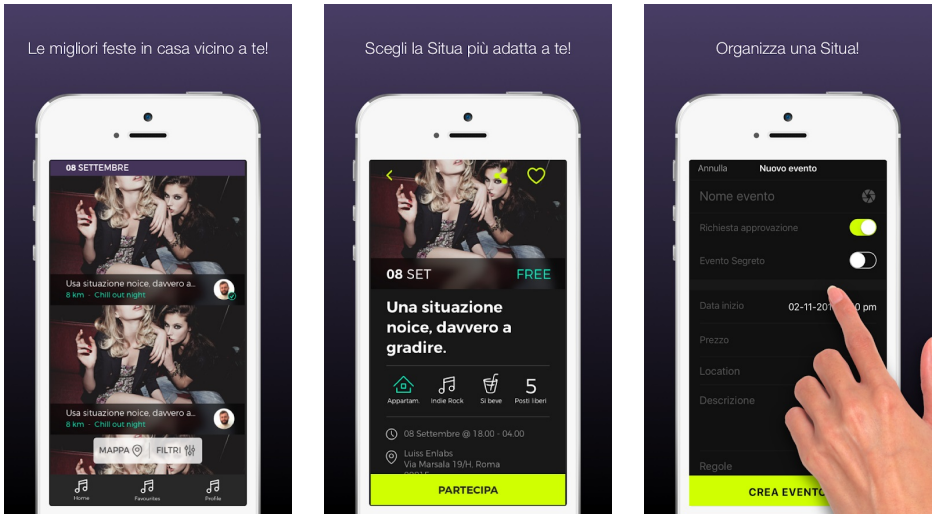


Figura 6.6: Alcuni screenshot dell'applicazione Situa

- *Plzcome Party Planner*: questa applicazione (Figura 6.7) permette la creazione di eventi di ogni genere mettendo a disposizione la possibilità di effettuare un sondaggio tra gli invitati per sapere quale giorno è più consono per fissare la festa. Inoltre, vi sono delle chat di gruppo utili per scambiarsi messaggi esclusivi dell'evento a cui si è stati invitati (Figura 6.8).

Figura 6.7: Plzcome Party App

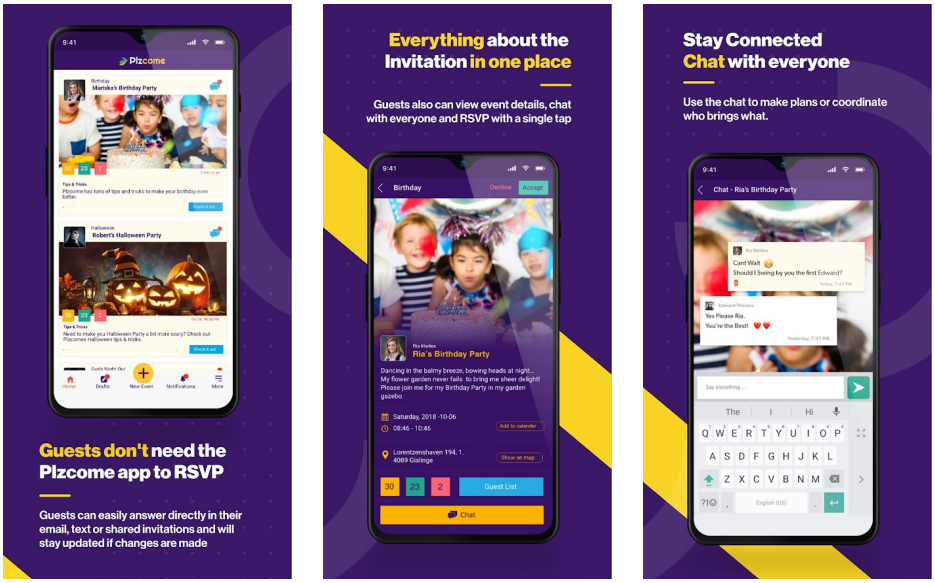


Figura 6.8: Alcuni screenshot dell'applicazione Plzcome Party App

Questa ricerca ci ha permesso di identificare le possibili "rivali" della nostra applicazione e di capire meglio quali possano essere i punti di forza e le debolezze della stessa. Infatti, delle applicazioni elencate nessuna gestisce gli eventi nell'era del Covid-19, come succede per la nostra. Un altro punto fondamentale è la gestione delle intolleranze e del menù che, nelle altre applicazioni, non vengono mai prese in considerazione ma che rappresentano una parte molto importante della nostra app.

7

Conclusioni

In questo elaborato di tesi si è visto il progetto HUP (Host Unforgettable Party), che consiste nella progettazione e realizzazione di un'app per la creazione di eventi nell'era del Covid-19.

Prima di tutto abbiamo fatto una panoramica sulla situazione attuale nel mondo e, in particolare, sulle misure che sta attuando l'Italia per fronteggiare l'emergenza sanitaria dovuta al Covid-19.

Nel secondo capitolo, invece, abbiamo distinto le varie tipologie di applicazioni introducendo l'ambiente di sviluppo Xamarin per le applicazioni ibride, come la nostra, ed elencandone i vantaggi e gli svantaggi nonchè diverse architetture per la condivisione del codice.

Dopo questi due capitoli introduttivi, abbiamo iniziato a parlare dell'applicazione vera e propria, fissandone i requisiti fondamentali, illustrando le funzionalità da implementare e i vincoli da rispettare, per poi passare alla fase di progettazione, in cui sono stati presentati dei diagrammi UML e dei mockup, utili per capire il funzionamento dell'app.

Nel Capitolo 4 è stata trattata l'implementazione; in particolare, è stata mostrata la struttura dell'app, illustrando e spiegando tutti i listati principali che la compongono.

In seguito, nel Capitolo 5, è stata presentata una guida all'utilizzo dell'applicazione, illustrando i diversi passaggi che un utente deve eseguire per poter creare un evento, modificarlo, invitare altri utenti e inserire le intolleranze, facendo attenzione, sempre grazie all'app, a rispettare tutte le normative vigenti riguardanti il Covid-19.

Infine, nel Capitolo 6, è stata introdotta la SWOT Analysis per poi applicarla al nostro progetto, utilizzando un giudizio critico per capire quali sono i punti di forza ma, soprattutto, i punti di debolezza della nostra applicazione.

Questo progetto è risultato essere molto utile per comprendere come sia possibile sviluppare un'applicazione ibrida per Android e a quante cose bisogna fare attenzione se si sviluppa un'app in questo momento storico particolare, con tutte le norme vigenti per la prevenzione del Covid-19. L'analisi delle app concorrenti ci ha fatto capire che nessuno prende in considerazione la situazione critica globale, probabilmente perchè la reputa una problematica passeggera, anche se ormai essa

affligge il mondo intero da più di un anno. La nostra applicazione invece, ne tiene conto e ne fa addirittura un punto di forza.

L'app, sebbene sia funzionante, può essere migliorata in diversi aspetti.

Per prima cosa si potrebbe aggiungere una chat privata dei singoli eventi; in questo modo gli invitati posso interagire tra di loro e con l'organizzatore; si potrebbe, inoltre, migliorare l'aspetto della privacy aggiungendo un meccanismo simile a quello delle amicizie di Facebook, con il quale potremmo avere in evidenza gli "amici" da invitare immediatamente, a differenza degli altri utenti che dovranno essere cercati singolarmente.

Riferimenti bibliografici

1. Miguel de icaza. *Wikipedia*, 2021.
2. Nat friedman. *Wikipedia*, 2021.
3. What is xamarin? *Xamarin*, 2021.
4. J. Bach. Xamarin. 2020.
5. M. de Icaza. Mono early history. *Mono.com*, 2003.
6. D. Hindrikes. Xamarin.forms projects. 2020.
7. D. S. Hui. The continuing 2019-ncov epidemic threat of novel coronaviruses to global health — the latest 2019 novel coronavirus outbreak in wuhan, china. *International Journal of Infectious Diseases*, 91, 2020.
8. Ionic. hybrid vs. native. *Ionic*, 2019.
9. Lab24. Cose che noi umani. *Il sole 24 ore*, 2020.
10. World Health Organization. Coronavirus disease (covid-19). *World Health Organization site*, 1, 2020.
11. C. Petzold. Creating mobile apps with xamarin.forms. 2015.
12. R. Re. Analisi swot: definizione, vantaggi e un esempio pratico. *Smart business lab.*
13. Net Market Share. Operating system market share. *netmarketshare.com*, 2016.
14. A. Tedeschi. Condivisione del codice: Shared assets project. *html.it.*
15. A. Tedeschi. Portable class libraries. *html.it.*
16. J. M. Wargo. Native, web, and cross-platform mobile apps all have their place. *Forrester*, 2016.

Ringraziamenti

Nel mio percorso formativo, il passaggio al grado superiore di istruzione, ha sempre rappresentato per me la scoperta di città sempre più grandi, la conoscenza di diverse realtà e nuove persone. È successo così dalle elementari alle medie, dalle medie alle superiori e anche dalle superiori all'università. Questo percorso mi ha condotto fin qui, e sono fiero di tutte le scelte che ho fatto fin'ora, è stato incredibile, arrivare all'università, in una città che, per me che provengo da un piccolo paese, sembra davvero immensa, è un'esperienza unica, non è stato sicuramente tutto rose e fiori, e non sono mancati momenti di smarrimento in cui mi sono messo più volte in discussione, ma se sono arrivato fino a qui è anche gran parte merito delle persone a me care che mi circondano.

Prima di tutto voglio ringraziare i miei genitori Clotilde e Domenico, mio fratello Alessandro e mia nonna Linda, per avermi sostenuto in ogni istante della mia vita, sostenendomi nei momenti difficili e festeggiando i miei successi come se fossero i loro. Voglio ringraziare inoltre tutta la mia famiglia, sia chi abita a pochi chilometri da me, sia chi è più distante e soprattutto a chi ora, mi protegge da lassù, perchè si sono sempre interessati ai miei progressi e hanno sempre creduto in me dandomi le motivazioni per andare avanti.

Un altro ringraziamento importantissimo lo voglio dedicare a chi è arrivato nella mia vita da poco più di 3 anni, alla mia ragazza Giulia che mi è stata accanto in ogni momento importante e mi ha sorretto nei momenti più difficili dandomi la forza e la determinazione necessaria per affrontare questa importante fase della mia vita.

Inoltre voglio fare un grande ringraziamento a tutti i miei amici, sia quelli che conosco da una vita, sia quelli che ho incontrato in questo viaggio. Senza di loro lo studio sarebbe stato molto più duro, e i momenti di svago meno divertenti. Faccio un sentito ringraziamento ai miei compagni di corso Alessandro, Angelo, Giorgio e Samuele, con cui ho affrontato molti progetti in questi anni; abbiamo dovuto affrontare numerose diffcoltà, ma insieme ce l'abbiamo sempre fatta.

Infine, un enorme ringraziamento va al mio relatore, il Professore Domenico Ursino. Lo ringrazio sentitamente per avermi impartito numerose nozioni, per avermi dato consigli riguardanti il mio futuro e per l'incredibile pazienza e disponibilità avuta nei miei confronti durante l'intero svolgimento del progetto.