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 di una social app a supporto degli amanti delle ricette e confronto tra le implementazioni Flutter e Xamarin

Design of a social app to support recipe lovers and comparison between the Flutter and Xamarin implementations

Relatore

Prof. Domenico Ursino

Candidato

Massimiliano Giovagnola

Anno Accademico 2020-2021

Indice

Elenco delle figure

Elenco dei listati

Introduzione

In un mondo così fortemente digitalizzato, lo smartphone è diventato un oggetto indispensabile nella vita di tutti i giorni. Tramite esso si possono effettuare molteplici operazioni tra cui, soprattutto, navigare su Internet per ottenere qualsiasi informazione che va dal semplice testo alle immagini, dalla navigazione per raggiungere una destinazione allo shopping online, e così via.

Negli ultimi anni si è cercato sempre più di racchiudere le funzionalità in applicazioni installabili sugli smartphone, in modo che l'utente possa raggiungerle in maniera semplice, veloce e intuitiva.

Quando si vuole creare una nuova applicazione, il programmatore può decidere di sviluppare l'app stessa in tre modalità differenti, ossia native, web app e ibride.

- Le applicazioni native sono sviluppate specificatamente per un sistema operativo, e ciò comporta l'implementazione di più versioni della stessa app per piattaforme differenti; al tempo stesso, offrono una maggiore velocità, affidabilità, reattività e, infine, l'accesso alle varie funzionalità dello smartphone.
- Una web app funziona come un sito web, senza differenze tra piattaforme, sviluppo e codice. Le web app consentono all'utente di raggiungere le varie funzionalità senza dover installare alcun applicativo aggiuntivo sul proprio dispositivo. Esse non permettono, però, da un lato di sfruttare al massimo le potenzialità del dispositivo, dall'altro di operare senza connessione ad Internet. Infine, esse risultano lente nell'esperienza d'uso dell'utente.
- Le applicazioni ibride, chiamate anche cross-platform, sono molto convenienti quando si vuole sviluppare un'app per più sistemi operativi, in quanto consentono di utilizzare la stessa implementazione. La manutenzione risulta più semplice anche se le performance, rispetto alle applicazioni native, risultano inferiori.

In questo elaborato si è scelto di progettare una social app a supporto degli amanti delle ricette, e di implementarla in un'app ibrida con l'obiettivo finale di pubblicarla sul Google Play Store, per sistemi Android, e, in futuro, su App Store, per iOS; in questo modo viene garantita la compatibilità dello stesso codice con le diverse piattaforme e, al tempo stesso, si hanno delle buone performance. Inoltre si è scelto, ai fini dello svolgimento di questo studio, di effettuare il testing dell'app solo su device Android poiché, per poter testare un'app su dispositivi iOS, è necessario

sia sottoscrivere un abbonamento annuale a pagamento, che l'utilizzo di un Mac, ossia del computer prodotto e venduto dall'azienda statunitense Apple.

Esistono molteplici framework per la creazione di applicazioni cross-platform; tra quelli attualmente più utilizzati abbiamo Flutter, di proprietà di Google, e Xamarin, di proprietà di Microsoft. Nella presente tesi tali framework verranno entrambi adottati per lo sviluppo dell'app in questione.

Obiettivo della fase preliminare del presente studio sarà l'acquisizione della conoscenza in merito ai due framework. In particolare, verranno analizzati gli ambienti di sviluppo, ossia Android Studio per Flutter e Visual Studio per Xamarin e, inoltre, saranno approfondite le varie componenti chiave necessarie per la programmazione.

Successivamente a questa fase preliminare, inizieranno l'analisi dei requisiti e la progettazione dell'app. Durante questa fase verranno stabiliti i requisiti funzionali e non dell'applicazione; successivamente, sarà definita la struttura dell'app e delle varie interazioni nelle operazioni da implementare. Verranno, poi, creati i vari mockup, che consentiranno di stabilire in maniera approssimativa l'interfaccia grafica che l'applicazione dovrà rispettare. Infine, verrà progettata la componente dati con la quale l'applicazione interagirà.

Terminata la fase di analisi e progettazione, il nostro studio proseguirà con l'implementazione della social app in Xamarin.

Terminata l'implementazione in Xamarin, la stessa applicazione verrà implementata tramite Flutter.

Una volta terminate le implementazioni, si procederà al confronto delle stesse, evidenziando le varie differenze sia nell'interfaccia grafica, sia nelle modalità di interazione con i dati, sia nel peso dei file APK, ossia i file che rendono possibile l'installazione dell'app nei sistemi Android.

L'ultima fase del nostro studio consisterà nel "tirare le somme" di tutto il lavoro svolto, evidenziando la sua importanza nel futuro come base di partenza per ulteriori sviluppi.

La presente tesi è strutturata come di seguito specificato:

- Nel Capitolo 1 verrà introdotto il framework Flutter, con particolare riferimento alle modalità di installazione. Successivamente verranno illustrate le operazioni per la creazione di un progetto; infine, verrà analizzata l'applicazione generata al termine di questa procedura.
- Nel Capitolo 2 verrà introdotto il framework Xamarin, con particolare riferimento nelle modalità di installazione. Successivamente verranno illustrate le operazioni per la creazione di un progetto: infine, verrà analizzata l'applicazione generata al termine di questa procedura.
- Nel Capitolo 3 verranno trattati sia l'analisi dei requisiti che la progettazione della social app. All'interno del capitolo verrà approfondita la struttura dell'app e verranno mostrati dei flowchart relativi ad alcune operazioni che l'applicazione dovrà svolgere. Saranno, poi, mostrati e spiegati i mockup utilizzati per definire l'interfaccia grafica dell'app; inoltre, verranno spiegate nel dettaglio le modalità sia di autenticazione dell'utente, sia di gestione dei dati dello stesso e delle ricette presenti nella social app, sia, infine, la gestione delle immagini.
- Nel Capitolo 4 verrà mostrata l'implementazione dell'applicazione in Xamarin, specificando nel dettaglio, il funzionamento delle varie componenti presenti.

- Nel Capitolo 5 verrà mostrata l'implementazione dell'applicazione in Flutter specificando nel dettaglio, il funzionamento delle varie componenti presenti.
- Nel Capitolo 6 verranno messe a confronto le implementazioni in Xamarin e Flutter. In particolare, verranno approfondite le differenze, sia nell'interfaccia grafica, sia nelle modalità di interazione con la componente dati, sia, infine, nella dimensione dei file APK.
- Nel Capitolo 7 verranno tratte le conclusioni in merito al lavoro svolto e si analizzeranno alcuni possibili sviluppi futuri.

1

Flutter

Nel maggio 2017 Google rilasciò un nuovo SDK basato su Dart, denominato Flutter. L'obiettivo principale della società di Mountain View era quello di poter sviluppare app cross-platform Android e iOS che fossero in grado di renderizzare all'estrema velocità dei 120 frame al secondo. Con il rilascio di Flutter 2.0 nel marzo 2021 si possono anche programmare web-app tramite Windows, macOS, Linux e ChromeOS.

1.1 IDE

L'IDE scelto da Google per programmare in Flutter è Android Studio, ovvero lo stesso ambiente di sviluppo per creare applicazioni native Android.

Tuttavia, per poter utilizzare questo SDK, oltre ad installare l'IDE precedentemente nominato, bisognerà effettuare qualche passaggio aggiuntivo.

1.1.1 Installazione (Windows)

La prima operazione da effettuare è quella di collegarsi al sito `flutter.dev` e cliccare sul pulsante "Get Started" in alto a destra (Figura 1.1).

Successivamente è necessario cliccare su "Windows" (Figura 1.2).

A questo punto è possibile scaricare il file .zip contenente l'SDK di Flutter cliccando sull'apposito pulsante (Figura 1.3).

Dopo aver aspettato la fine del download, estrarre il contenuto in una cartella diversa da `C:\ProgramFiles\` che richiederebbe i privilegi dell'amministratore; a questo punto si estrae il file .zip, ad esempio, in `C:\src\flutter\` (che, d'ora in poi, verrà chiamata "cartella di esempio").

Per poter eseguire comandi Flutter dal "Promt dei comandi" di Windows, si dovrà aggiungere Flutter alla variabile d'ambiente `PATH` : per fare ciò bisognerà ricercare "env" nello Start menu di Windows e cliccare su "Modifica variabili di ambiente per l'account". Una volta aperta questa videata, sarà necessario selezionare la variabile `Path` e cliccare su "Modifica" (Figura 1.4).

A questo punto si seleziona la prima riga vuota disponibile e si scrive, come nella "cartella di esempio" sotto riportata, `C:\src\flutter\bin`. Dopo di ciò si preme "OK" (Figura 1.5).

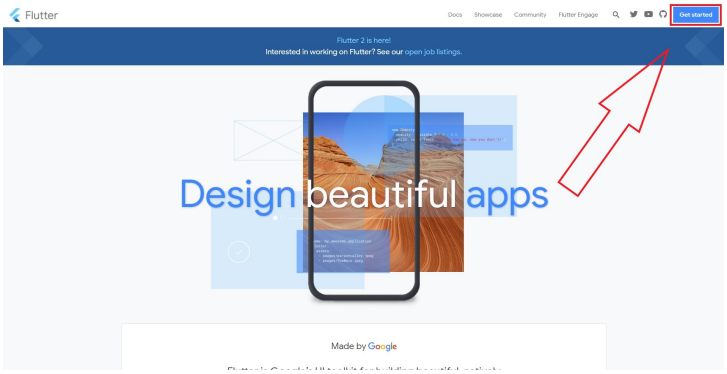

Figura 1.1. Home Page di `flutter.dev`

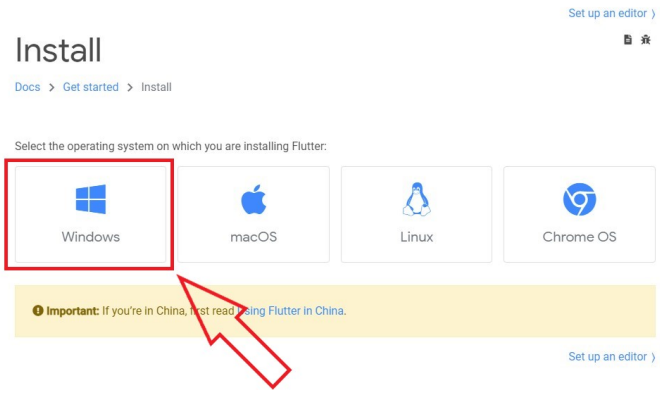

Figura 1.2. Pulsante Windows

Figura 1.3. Pulsante per il download SDK Flutter

Se la variabile `Path` non esiste, cliccare sul pulsante "Nuova" (Figura 1.6).

Dopo di ciò si inserisce nel campo "Nome variabile" il valore "Path" e, nella cella sottostante, si scrive `C:\src\flutter\bin` (come nella "cartella di esempio" sotto riportata) e si preme "OK" (Figura 1.7).

Arrivati a questo punto si può aprire il terminale (cercando "cmd" nello Start menu). Dopo di ciò:

- se si è aggiunto Flutter alle variabili di sistema, si esegue il comando `flutter doctor`;

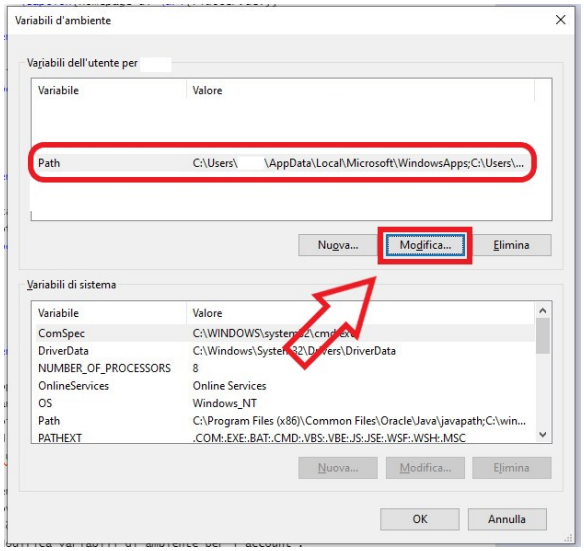

Figura 1.4. Finestra per la modifica della variabile `Path`

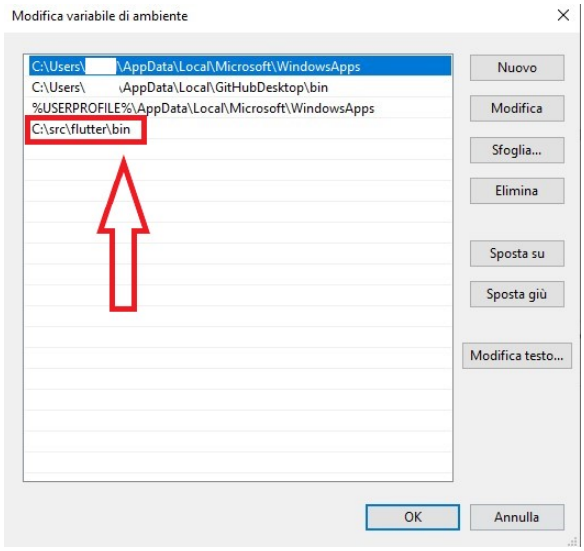

Figura 1.5. Inserimento dei parametri nella variabile `Path`

- se non lo si è fatto, si naviga tramite terminale nella cartella dove si trova la SDK estratta di Flutter, si entra nella cartella `bin` e si esegue il comando precedentemente indicato.

Tramite il comando `flutter doctor` si controlla lo stato di Flutter stesso e, se ci dovessero essere degli errori, vengono mostrate le anomalie contrassegnandole con una X e un testo in grassetto.

Solitamente, al presentarsi di questi imprevisti, verrà suggerita una possibile

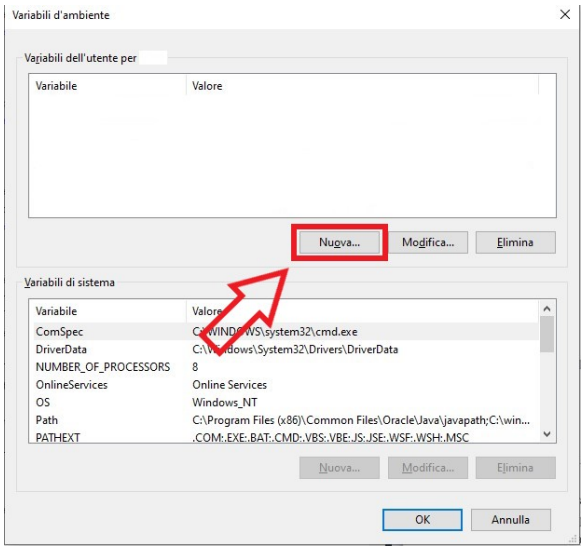

Figura 1.6. Pulsante per l'aggiunta della variabile `Path`

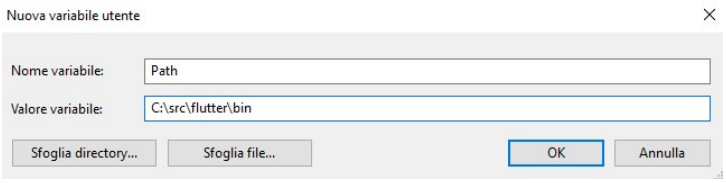

Figura 1.7. Aggiunta della variabile `Path`

soluzione.

Se non si trovano dei problemi, si può procedere con l'avvio di Android Studio, si clicca in basso "Configure" e poi "Plugins" (Figura 1.8).

Arrivati a questo punto, si dovrà cercare "Flutter" nel Marketplace ed si installerà il relativo plugin.

Se l'operazione verrà eseguita con successo, il risultato sarà simile a quello mostrato in Figura 1.9.

1.1.2 Creazione di un nuovo progetto Flutter

Arrivati a questo punto si può procedere con la creazione di nuovo progetto Flutter. Per fare ciò si dovrà aprire Android Studio e cliccare sul pulsante "Create New Flutter Project", come mostrato nella figura sottostante (Figura 1.10).

Alla successiva videata selezionare "Flutter Application" e premere "Next" (Figura 1.11).

Nella finestra successiva scrivere nella prima cella il nome desiderato per il nuovo progetto che, in questo esempio, è `flutter_app`.
Nella seconda cella indicare il percorso contenente la SDK di Flutter (se si è usata la "cartella di esempio" scrivere `C:\src\flutter`).

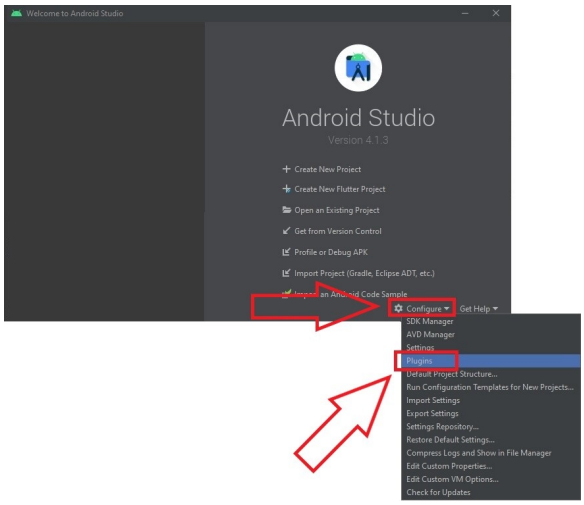

Figura 1.8. Plugin in Android Studio

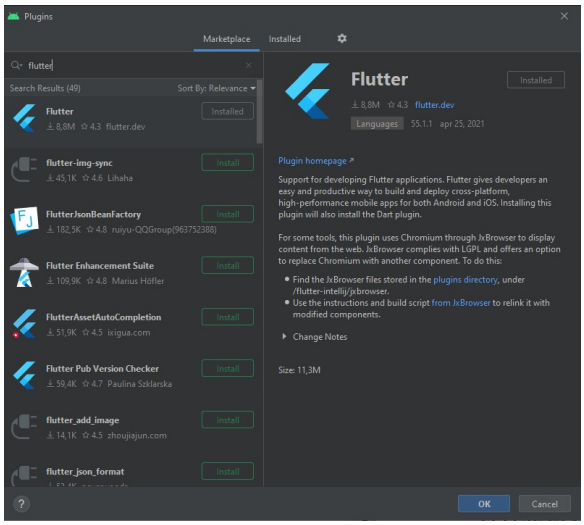

Figura 1.9. Plugin Flutter installato in Android Studio

Nella terza la cartella di destinazione del progetto.
Nell'ultima una descrizione. Compilati i seguenti campi premere "Next", come indicato in Figura 1.12.

Impostare il nome del pacchetto desiderato e continuare cliccando "Finish" (Figura 1.13).

Per verificare la corretta creazione del progetto basterà eseguire il codice generato automaticamente.

Per fare ciò si seleziona, nella barra degli strumenti, un emulatore Android, si aspetta la sua apertura e operatività e, successivamente, si preme il tasto "Play" verde (Figura 1.14).

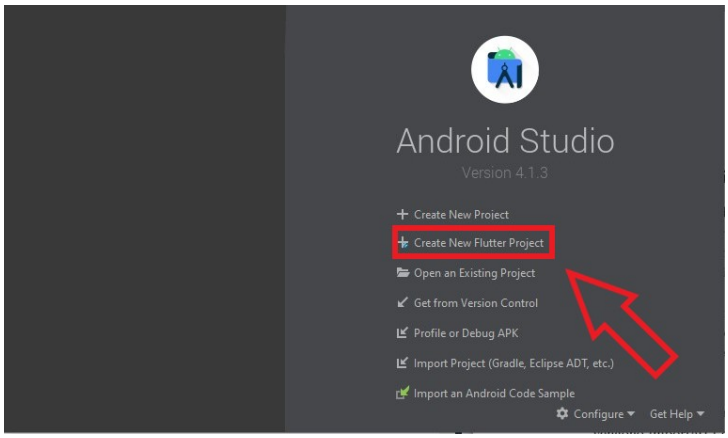

Figura 1.10. Pulsante per la creazione di un nuovo progetto Flutter

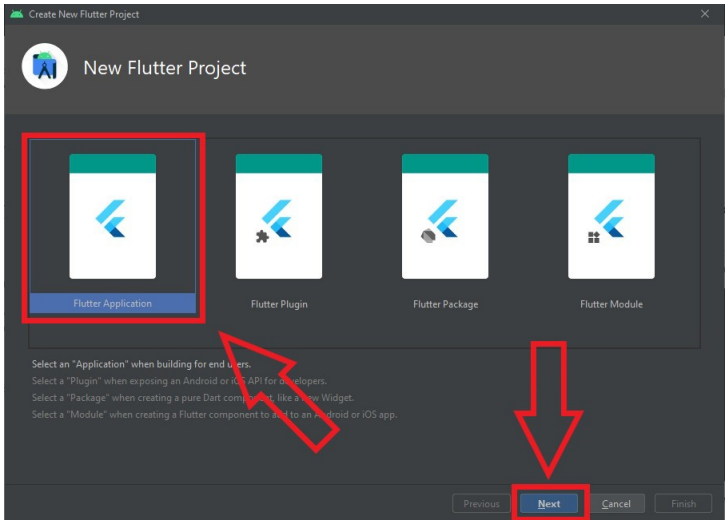

Figura 1.11. Selezione di Flutter Application

Se durante la compilazione e l'installazione non si sono verificati errori, il risultato mostrato dall'emulatore sarà simile a quello mostrato in Figura 1.15.

L'applicazione (d'ora in poi "app d'esempio" o "applicazione d'esempio") mostrata nella figura precedente è piuttosto semplice nel suo funzionamento, al tocco del pulsante "+" in basso a destra verrà incrementato il valore del numero intero presente al centro della schermata. Il codice corrispondente è riportato nel Listato 1.1.

```
import 'package:flutter/material.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
```

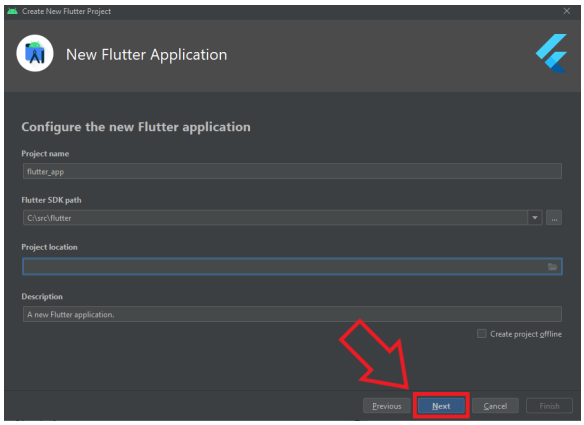

Figura 1.12. Configurazione del nuovo progetto Flutter

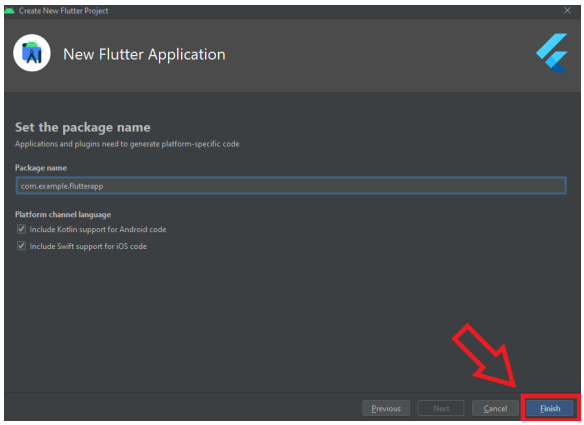

Figura 1.13. Fine della procedura di creazione del progetto

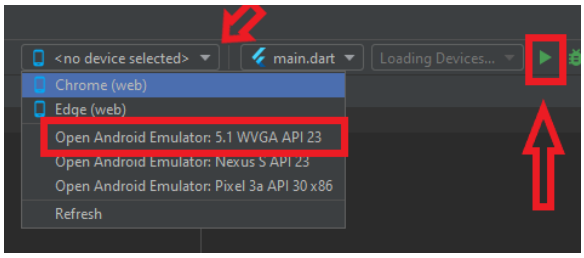

Figura 1.14. Esecuzione dell'app

```
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}
```

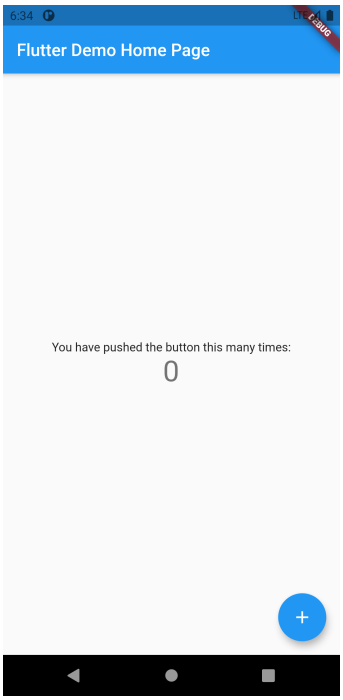

Figura 1.15. App iniziale

```
class MyHomePage extends StatefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);
  final String title;

  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              'You have pushed the button this many times:',
            ),
            Text(
              '$_counter',
              style: Theme.of(context).textTheme.headline4,
            ),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: Icon(Icons.add),
      ),
    );
```

```
    }
}
```

Listato 1.1. Applicazione d'esempio in Flutter

1.2 Interfaccia utente

Se si è soliti programmare in Xamarin e/o Android nativo, si potrebbe trovare qualche difficoltà nel leggere il codice riportato nel paragrafo precedente poiché Flutter ha una logica di programmazione diversa rispetto alla programmazione tipica in C# o Java.

Nel metodo `main()` è presente soltanto una riga di codice: `runApp(MyApp())`.

La funzione `runApp()` imposta il widget passato come parametro come radice dell'app.

1.2.1 Widgets

In Flutter non esistono file di layout simili a Xamarin (file `.xaml`) o nativo Android (file `.xml`), ma la User Interface viene definita attraverso l'uso di widget nei file `.dart`, gli stessi file dove possono essere scritte, ad esempio, classi e metodi.

Un widget non necessario, ma che è buona norma usare è `MaterialApp`, che costruisce una serie di widget utili alla radice dell'app come, ad esempio, `Navigator` che gestisce lo stack di widget identificati da stringhe (chiamate route). `Navigator` consente di spostarsi con estrema fluidità tra una schermata e l'altra dell'applicazione.
Uno dei parametri di MaterialApp è `theme`, al quale viene associato il widget `ThemeData` che definisce il tema dell'intera applicazione.

Come si può notare nel codice riportato precedentemente, le classi presenti sono sottoclassi rispettivamente di `StatelessWidget` e `StatefulWidget`, le quali implementano la funzione `build()`; in `build()` viene descritta la User Interface dell'applicazione.

Per spiegare la differenza tra `StatelessWidget` e `StatefulWidget` occorre prima introdurre il concetto di `state` (stato).

Lo *stato* è un'informazione che può essere letta quando viene eseguito il metodo `build()` e che potrebbe cambiare durante l'esecuzione del widget.

StatelessWidgets

Come facilmente intuibile, uno `StatelessWidget` è un widget che non possiede uno stato mutabile; esso descrive parti dell'applicazione statiche (che non cambiano mai) e che dipendono soltanto dalla configurazione stabilita nel metodo `build()`.

In "applicazione di esempio" lo `StatelessWidget` è usato per definire il tema.

StatefulWidget

Uno `StatefulWidget` è un widget che ha uno stato che potrebbe variare durante l'esecuzione del widget stesso e che richiede un aggiornamento della User Interface.

In "applicazione d'esempio", quando si clicca sul pulsante "+", posizionato in basso a destra, si invoca il metodo `_incrementCounter()` che richiama, a sua volta, il metodo `setState()`, dove viene fatto incrementare di 1 il valore della variabile `_counter`.

Tramite medodo `setState()` si notifica al framework che lo stato è cambiato e che, quindi, è necessario effettuare il re-building della User Interface. In questo caso, nel caso di "app d'esempio", verrà aggiornato il valore del widget *Text*, il quale mostra il valore della variabile `_counter`.

Si sottolinea che `setState()` aggiorna la User Interface con i valori in memoria delle variabili presenti in quel momento, anche se queste ultime non sono indicate nelle parentesi graffe di questa funzione.

1.2.2 Layout

I widget di layout sono essenziali affinché vengano mostrati in maniera corretta altri widget come *Text, Image, FloatingActionButton, TextField, etc.*. Le caratteristiche di alcuni di questi sono di seguito specificati:

- *Scaffold*: è un widget che fornisce di default un banner, un colore di background e offre supporto a molti widget come, ad esempio, i *FloatingActionButton* (in "app d'esempio" sono sufficienti quattro brevi righe di codice per aggiungerlo e impostarlo).
- *SafeArea*: è un widget che inserisce il widget figlio con un padding sufficiente per evitare intrusioni dal sistema operativo (ad esempio nell'iPhone 12, *SafeArea* fornisce abbastanza padding per evitare che il contenuto del widget figlio finisca al di sotto della tacca del gruppo fotocamera anteriore-altoparlante).
- *Column*: è un widget che mostra i widget figli verticalmente.
- *Row*: è un widget che mostra i widget figli orizzontalmente.
- *Container* è un widget che imposta dei parametri al widget figlio come, ad esempio, del padding, posizionamento, dimensione etc.
- *SizedBox*: è un widget con una dimensione specificata dal programmatore.

2

Xamarin

Xamarin, originariamente creato dall'omonima azienda, è un framework per lo sviluppo di app native e cross-platform basato su C#. Esso si basa sul progetto open source Mono, offrendo pieno supporto alle piattaforme Android, iOS e Windows Phone. Con l'acquisizione effettuata dall'azienda informatica Microsoft nel 2016, divenne possibile, agli sviluppatori e agli studenti, utilizzare gratuitamente Xamarin nella programmazione tramite macchine Windows, macOS e, più recentemente, Linux.

2.1 IDE

Nel 2013 Xamarin scelse il rilascio come ambiente di sviluppo "Xamarin Studio", disponibile a pagamento per sistemi Windows e macOS.

Con l'acquisto di Xamarin da parte della società di Redmond ciò cambiò poiché, per Windows, venne deprecato l'IDE originario e sostituito con "Xamarin for Visual Studio" mentre, per macOS, venne rinominato "Visual Studio for Mac".

2.1.1 Installazione (Windows)

La prima operazione da effettuare è quella di collegarsi su `https://visualstudio.microsoft.com/it/xamarin/` e cliccare sul pulsante "Download for Windows" (Figura 2.1). Viene mostrato un menù a discesa dove è necessario selezionare la versione di Visual Studio desiderata; nell'esempio, supponiamo di scaricare la versione gratuita "Community".

Finito il download è necessario eseguire il file `.exe` scaricato, autorizzando l'applicativo ad apportare modifiche al dispositivo. Viene a questo punto visualizzata una schermata nella quale viene richiesto di installare "Visual Studio Installer", ossia un'applicazione che consente l'installazione di Visual Studio. Per avanzare con la procedura si clicca sul pulsante "Continua" (Figura 2.2).

Dopo aver atteso la fine del download e l'installazione di "Visual Studio Installer", viene mostrata una finestra nella quale è possibile selezionare le componenti

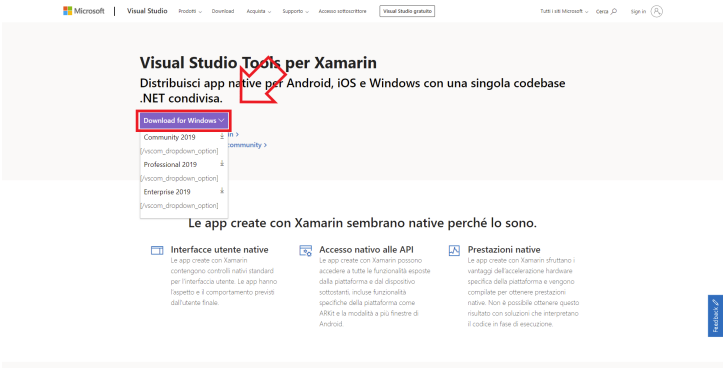

Figura 2.1. Pulsante per il download in `visualstudio.microsoft.com/it/xamarin/`

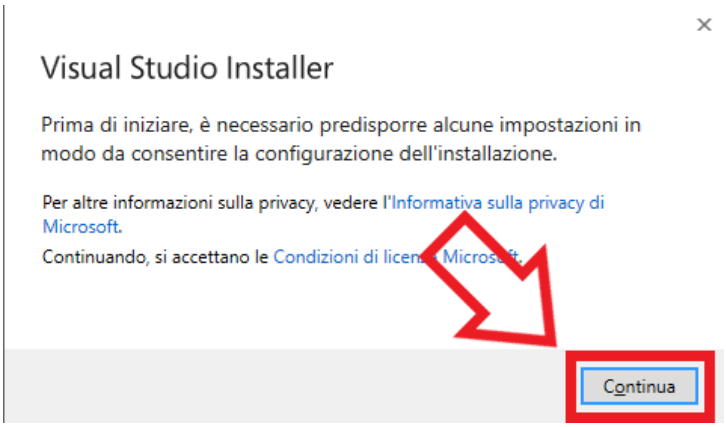

Figura 2.2. Finestra per consentire l'installazione di "Visual Studio Installer"

aggiuntive che si possono integrare a Visual Studio. Nella finestra, scorrendo verso il basso, si spunta l'opzione "Sviluppo di applicazioni per dispositivi mobili con .NET" e si preme il pulsante "Installa" (Figura 2.3).

Una volta terminati download e installazione di Visual Studio, si attiva una finestra nella quale viene richiesto di autenticarsi. Per saltare questa operazione si scelga "Non ora, forse in seguito" (Figura 2.4).

Alla successiva finestra è possibile scegliere il tema colori di proprio piacimento; a questo punto si preme sui "Avvia Visual Studio" (Figura 2.5) e si dovrà attendere l'avvio dell'IDE.

2.1.2 Creazione di un nuovo progetto Xamarin

Per poter iniziare la procedura di creazione di un nuovo progetto, per lo sviluppo di una nuova app cross-platform Xamarin, è necessario selezionare "Crea un nuovo progetto" (Figura 2.6).

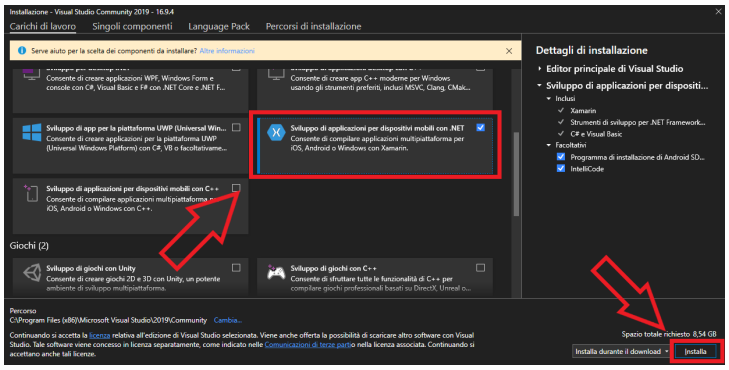

Figura 2.3. Selezione di "Sviluppo di applicazioni per dispositivi mobili con .NET"

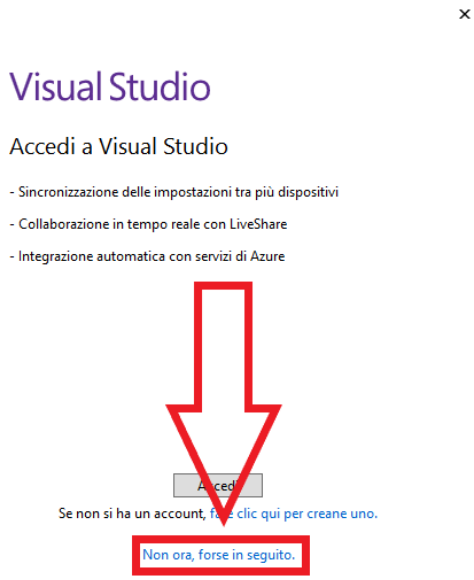

Figura 2.4. Selezione di "Non ora, forse in seguito"

A questo punto si scrive "Xamarin.Forms", nella casella di testo presente nella finestra, si seleziona "App per dispositivi mobili (Xamarin.Forms)" e si preme "Avanti", come riportato in Figura 2.7.

Nella finestra successiva è necessario impostare il nome del progetto, nella prima casella di testo, il percorso, nella seconda, e, nella terza, il nome della soluzione. Una volta inseriti questi dati, per poter proseguire, si clicca su "Crea". (Figura 2.8).

È ora necessario scegliere il modello di partenza dell'applicazione; in questo esempio si utilizza il modello denominato "Vuota". Successivamente si clicca sul pulsante "Crea" (Figura 2.9) per poter generare il progetto.

Per verificare la corretta creazione del progetto Xamarin, basterà eseguire il

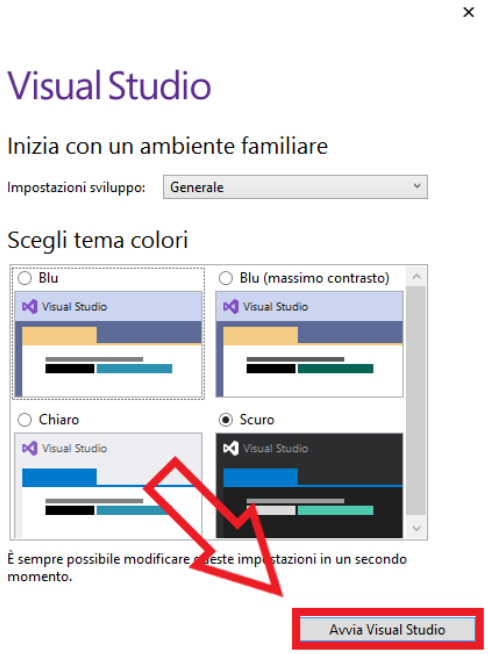

Figura 2.5. Fine della procedura di installazione

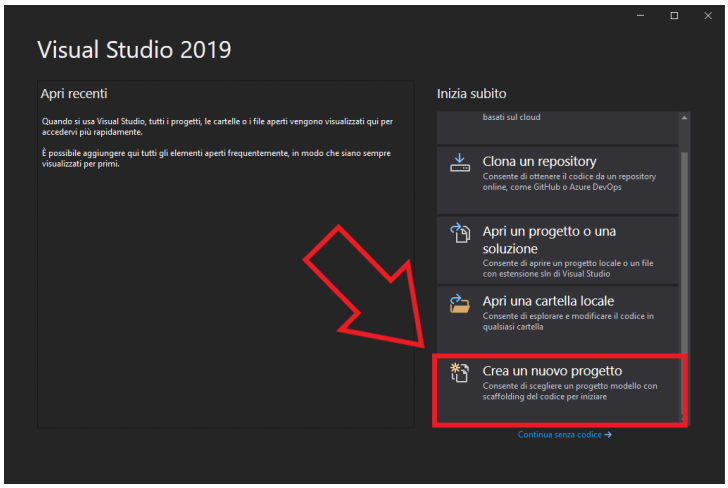

Figura 2.6. Selezione di "Crea un nuovo progetto"

codice generato automaticamente.

Per fare ciò si clicca sul tasto "Play" verde, posizionato nella barra degli strumenti, al fine di avviare l'esecuzione tramite un emulatore Android (Figura 2.10).

Il risultato mostrato dall'emulatore sarà simile a quello di Figura 2.11, se non si sono verificati errori durante la compilazione e l'installazione.

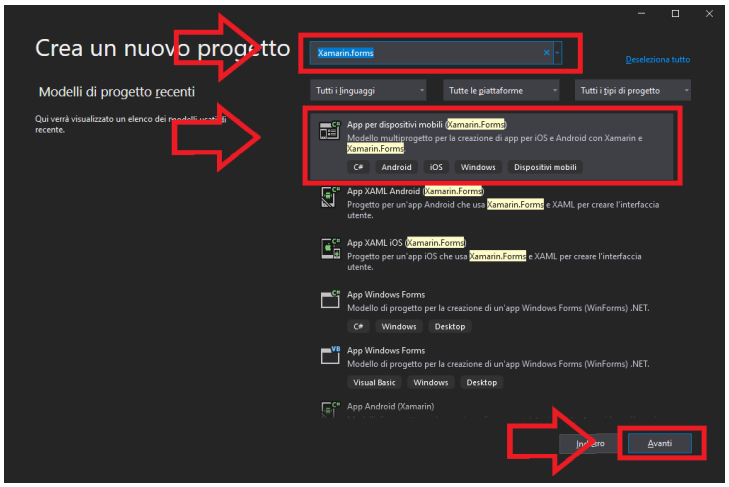

Figura 2.7. Selezione di "Xamarin.Forms"

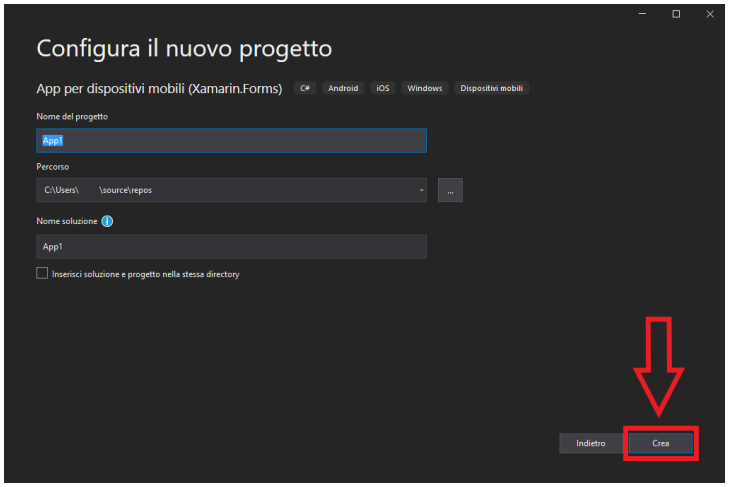

Figura 2.8. Scelta del nome del progetto Xamarin

L'applicazione (d'ora in poi "app d'esempio" o "applicazione d'esempio") riportata nella Figura 2.11 mostra un testo. I codici corrispondenti sono riportati nei Listati 2.1, 2.2, 2.3 e 2.4.

```
<?xml version="1.0" encoding="utf-8" ?>
<Application xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="App1.App">
    <Application.Resources>

    </Application.Resources>
</Application>
```

Listato 2.1. `App.xaml` dell'"app d'esempio"

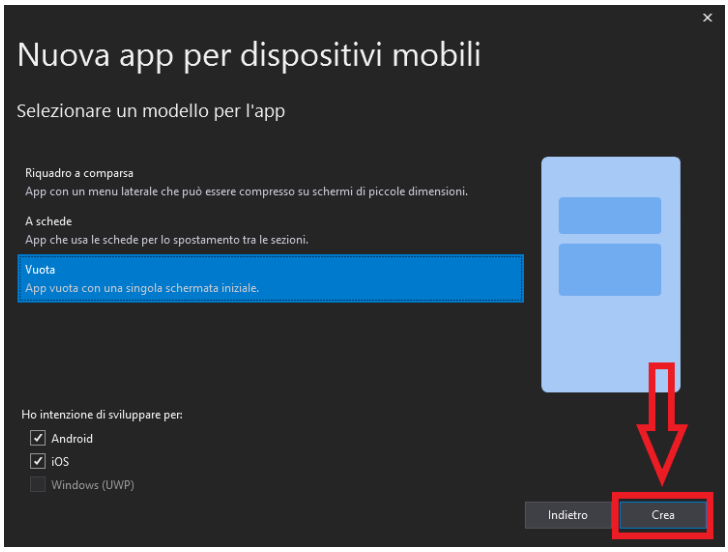

Figura 2.9. Fine della procedura di creazione del progetto Xamarin

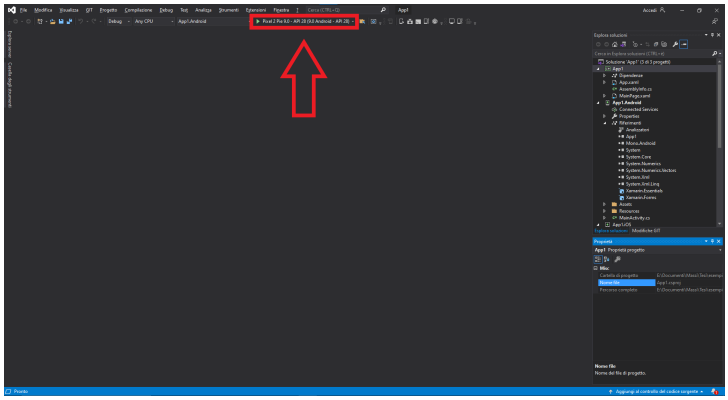

Figura 2.10. Esecuzione dell'app

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

namespace App1
{
    public partial class App : Application
    {
        public App()
        {
            InitializeComponent();

            MainPage = new NavigationPage(new MainPage());
        }

        protected override void OnStart()
        {
        }

        protected override void OnSleep()
        {
        }
```

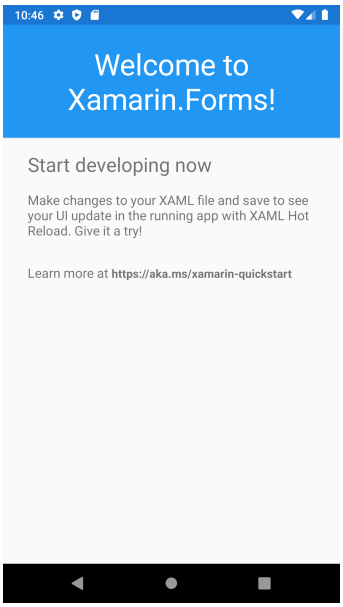

Figura 2.11. Esecuzione dell'app

```
        protected override void OnResume()
        {
        }
    }
}
```

Listato 2.2. `App.xaml.cs` dell'"app d'esempio"

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="App1.MainPage">

    <StackLayout>
        <Frame BackgroundColor="#2196F3" Padding="24" CornerRadius="0">
            <Label Text="Welcome to Xamarin.Forms!"
            HorizontalTextAlignment="Center" TextColor="White" FontSize="36"/>
        </Frame>
        <Label Text="Start developing now" FontSize="Title" Padding="30,10,30,10"/>
        <Label Text="Make changes to your XAML file and save to see your UI update
        in the running app with XAML Hot Reload. Give it a try!" FontSize="16" Padding="30,0,30,0"/>
        <Label FontSize="16" Padding="30,24,30,0">
            <Label.FormattedText>
                <FormattedString>
                    <FormattedString.Spans>
                        <Span Text="Learn more at "/>
                        <Span Text="https://aka.ms/xamarin-quickstart" FontAttributes="Bold"/>
                    </FormattedString.Spans>
                </FormattedString>
            </Label.FormattedText>
        </Label>
    </StackLayout>

</ContentPage>
```

Listato 2.3. `Mainpage.xaml` dell'"app d'esempio"

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;

namespace App1
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
    }
}
```

Listato 2.4. `Mainpage.xaml.cs` dell'"app d'esempio"

2.2 Codice

I listati sopra riportati mostrano file con estensione `.xaml` ed altri con estensione `.xaml.cs`.

XAML, abbreviazione di eXtensible Application Markup Language, è un linguaggio proprietario Microsoft di markup basato su XML e serve a descrivere la User Interface nelle applicazioni Xamarin. Durante la creazione di un file `.xaml` verrà associato un file con estensione `.xaml.cs`, dove sarà possibile scrivere del codice per gestire l'interfaccia utente.

Sia `App.xaml` (Listato 2.1) che `App.xaml.cs` (Listato 2.2) contribuiscono ad una classe chiamata `App` che deriva da `Application`.

`Mainpage.xaml` (Listato 2.3) e `Mainpage.xaml.cs` (Listato 2.4) contribuiscono alla classe `ContentPage`, e tramite loro, viene definito l'effettivo contenuto visivo nell'"app d'esempio".

Nelle primissime righe di entrambi i file `.xaml` si trovano due *namespace* che fanno riferimento ad altrettanti *URI* (`xmlns="http://xamarin.com/schemas/2014/forms"` e `xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"`); con questa organizzazione si definisce che tutti i tag scritti nello XAML senza prefisso si riferiscono a classi di Xamarin.Forms.

Si noti che nel secondo namespace è presente il prefisso "`x`" poichè esso specifica che l'*URI* definito è di proprietà di Microsoft.

Importantissima è la riga di codice successiva, ossia `x:Class="App1.NomeFile"`, perchè, tramite essa, viene specificata un nome di una classe *.NET* definita nel namespace `App1`; tramite questa riga si crea un "ponte" tra il file con estensione `.xaml` e il file di code-behind con estensione `.xaml.cs`.

Nei file `.xaml.cs` è presente la keyword `partial` la quale indica che la classe corrispondente potrebbe essere definita su più file diversi; in uno di questi ultimi, ad esempio `Mainpage.xaml.g.cs`, è presente sia la definizione mancante della classe `Mainpage` in `Mainpage.xaml.cs` che il metodo `InitializeComponent()` chiamato dal costruttore di `Mainpage`.

La funzione `InitializeComponent()` richiama il metodo `LoadFromXaml()` e quest'ultimo provvede sia a inizializzare tutti gli oggetti definiti nel file `.xaml`, sia a connetterli con le eventuali relazioni padre-figlio, sia ad associare i gestori eventi definiti nel codice agli eventi impostati nel file `.xaml`, sia, infine, a impostare la struttura ad albero risultante degli oggetti come contenuto della pagina.

Quando l'applicazione sarà eseguita verrà istanziata la classe `App`, che eseguirà il metodo `InitializeComponent()` di `App.xaml.cs` e, successivamente, `MainPage = new NavigationPage(new MainPage())` che consentirà la navigazione alla `Mainpage` la quale, a sua volta, mostrerà il suo contenuto dopo aver eseguito la funzione `InitializeComponent()`.

Nella classe `App.xaml.cs` si trovano, anche, i metodi provenienti dalla classe `Application`; questi sono:

- `OnStart()`: chiamato all'avvio dell'applicazione.
- `OnSleep()`: chiamato ogni volta che l'app passa in background.
- `OnResume()`: chiamato alla ripresa dell'applicazione, dopo il passaggio dell'app.

A differenza di quanto avviene per le app native Android, non esiste un metodo per la terminazione dell'app la quale, a meno di un arresto anomalo, verrà eseguita dal metodo `OnSleep()`.

2.3 Layout

Se in Flutter i widget definiscono la User Interface, in Xamarin questa viene definita con l'uso di Page, Layout e View.

Page è il container principale e rappresenta la generica schermata che può contenere vari Layout per poter posizionare e dimensionare i contenuti. I tipi principali di Page sono:

- *ContentPage*: consente la visualizzazione di un solo un contenitore come StackLayout o ScrollView (Figura 2.12).
- *MasterDetailPage*: gestisce due riquadri di informazioni. Il primo riquadro, chiamato *Master*, viene impostato con delle proprietà "generali" che verranno mostrate all'utente sotto forma di elenco. Il secondo riquadro, chiamato *Detail*, viene costruito in relazione all'elemento *Master* che l'utente ha selezionato (Figura 2.13).
- *NavigationPage*: gestisce lo spostamento tra le altre pagine attraverso un'organizzazione a stack (Figura 2.14).
- *TabbedPage*: consente la navigazione all'interno delle schede della pagina (Figura 2.15).
- *CarouselPage*: consente la navigazione tra la schermata principale (padre) e le schermate figlie tramite lo scorrimento del dito (Figura 2.16).

Gli oggetti "Views" sono posizionati e dimensionati nella classe Layout; i principali tipi di layout in Xamarin sono di seguito riportati (Figura 2.17):

- *StackLayout*: posiziona gli elementi figlio in una singola riga che può essere orientata orizzontalmente o verticalmente.
- *AbsoluteLayout*: posiziona gli elementi figlio in corrispondenza delle posizioni assolute.
- *RelativeLayout*: usa dei vincoli (*Costraints*) per creare il layout dei relativi elementi figlio.
- *Grid*: dispone le visualizzazioni in righe e colonne.

Figura 2.12. Esempio di ContentPage

Figura 2.13. Esempio di MasterDetailPage

Figura 2.14. Esempio di NavigationPage

Figura 2.15. Esempio di TabbedPage

Figura 2.16. Esempio di CarouselPage

- *ScrollView*: supporta lo scorrimento se richiesto del relativo contenuto.
- *Frame*: contiene un singolo elemento figlio, con alcune opzioni di framing.
- *ContentView*: contiene un singolo elemento figlio.

In Xamarin gli oggetti "Views" sono tutti gli elementi di controllo, visibili ed interattivi in una pagina.

Una View può essere un pulsante, una casella di testo o una lista e possiede diverse proprietà che consentono la definizione del contenuto, del font, del colore etc.

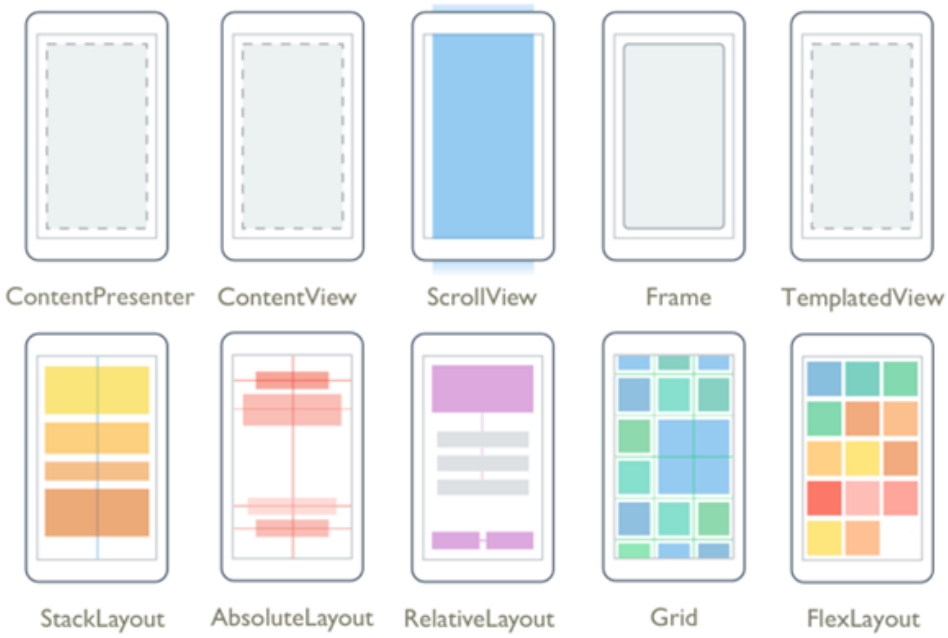

Figura 2.17. Alcuni layout in Xamarin.Forms

3

Analisi dei requisiti e progettazione dell'app

In questo capitolo illustreremo l'analisi dei requisiti relativi a Knife&Spoon, la social app a supporto degli amanti delle ricette oggetto della presente tesi. Successivamente, procederemo con la fase di progettazione dell'app stessa.

3.1 Analisi dei requisiti

3.1.1 Descrizione del progetto

L'obiettivo che si intende raggiungere con *Knife&Spoon* è un'applicazione che sia in grado di gestire tre tipi diversi di utenti offrendo, a ciascuno di essi, diverse funzionalità.

Il primo tipo di utente che si vuole gestire è quello anonimo, ossia un utente che non effettua il login all'applicazione tramite Google, e che può solo visualizzare e ricercare le ricette presenti nel database.

La seconda tipologia di utente che si vuole gestire è quello registrato ma "non-admin", che può visualizzare, ricercare, impostare come preferito e pubblicare le ricette che dovranno essere revisionate dall'utente "admin".

L'ultima tipologia di utente è l'"admin" che, oltre a possedere tutte le funzionalità dell'utente "non-admin", può revisionare, approvare o eliminare una ricetta appena pubblicata da un utente registrato.

Una ricetta approvata può essere visualizzata da tutti, gli utenti indipendentemente dalla tipologia a cui appartengono.

3.1.2 Requisiti

I requisiti dell'applicazione che si vuole sviluppare, descritti in questa regione, si suddividono in funzionali e non funzionali; i primi rappresentano l'insieme delle funzionalità che si devono implementare nell'applicazione mentre i secondi sono costituiti dall'insieme dei vincoli e delle regole, sia di tipo realizzativo che tecnico, che essa deve rispettare.

Requisiti funzionali

L'applicazione che si sviluppa deve garantire le seguenti funzionalità:

- registrazione di un nuovo utente tramite Google;
- login di un utente già registrato tramite Google;
- login di un utente anonimo;
- inserimento di una nuova ricetta;
- visualizzazione di una ricetta;
- aggiunta o rimozione di una ricetta dai preferiti;
- visualizzazione delle ricette preferite;
- cambio dell'immagine del profilo;
- visualizzazione delle ricette da approvare;
- approvazione di una ricetta.

Requisiti non funzionali

L'applicazione che si sviluppa deve garantire i seguenti requisiti non funzionali:

- interfaccia utente user-friendly e moderna;
- fluidità e reattività;
- affidabilità;
- gestione dei dati in remoto.

3.2 Progettazione

3.2.1 Struttura dell'app

La struttura dell'applicazione specifica come sono organizzate le informazioni presenti; in altri termini, indica quali sono i "passi" per raggiungere una determinata schermata da un qualsiasi punto dell'applicazione.

L'obiettivo finale è quello di aiutare l'utente nell'utilizzo dell'applicazione rendendo più facile la ricerca di una particolare sezione dell'applicativo.

La struttura dell'app è riportata nella Figura 3.1.

3.2.2 Flowchart

I grafici riportati di seguito illustrano alcune funzionalità dell'applicazione e, più in particolare, il login tramite Google e l'inserimento di una nuova ricetta.

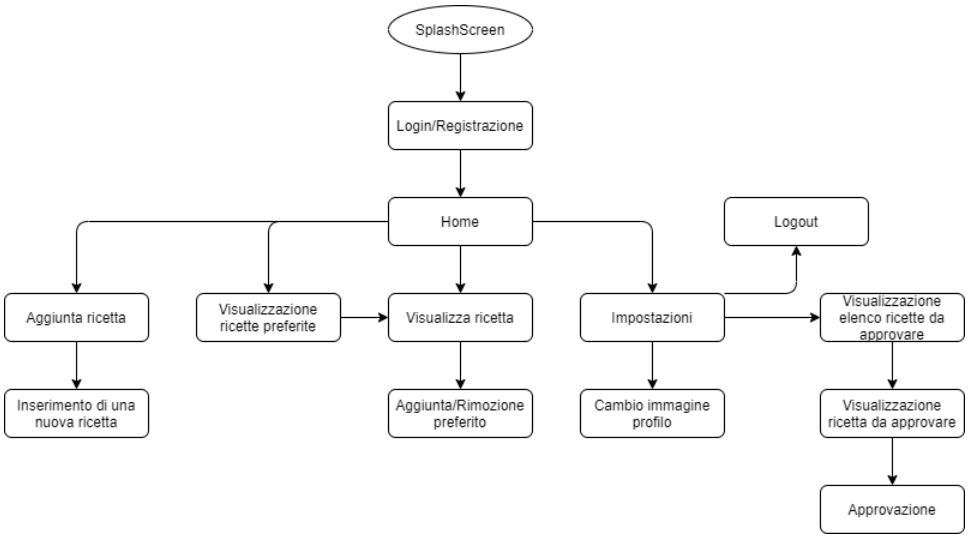

Figura 3.1. Struttura dell'applicazione

Login tramite Google

Le attività che vengono svolte per gestire il login dell'applicazione tramite Google vengono riportate in Figura 3.2. In particolare, l'utente che si trova nella schermata di accesso e clicca sul pulsante del login tramite Google può selezionare l'account di sua preferenza. Una volta effettuato l'accesso tramite il servizio di autenticazione, l'applicazione memorizza i dati necessari per accedere automaticamente in futuro; se si verifica un errore, viene mostrato un messaggio ed occorre effettuare di nuovo la procedura di accesso. Se non si sono verificati errori e se l'utente non ha mai completato la procedura di registrazione, viene mostrata una schermata in cui si avvisa che è necessario inserire il proprio username. A questo punto l'utente clicca sul pulsante che consente di proseguire; se il nome utente scelto risulta già presente nel database, viene mostrato un messaggio con il quale l'utente è invitato a variare il proprio username.

Inserimento di una nuova ricetta

Le attività che vengono svolte per l'inserimento di una nuova ricetta sono riportate in Figura 3.3. In particolare, l'utente registrato che si trova nella schermata della home page, cliccando sul menù e, successivamente, sul pulsante "Aggiungi", accede alla videata per l'inserimento di una nuova ricetta.

Se l'utente registrato compila tutti i campi necessari e inserisce una fotografia del piatto, l'applicazione controlla se questi sono corretti; se non si rilevano errori, la ricetta viene caricata nel database online in attesa di approvazione da parte di un "admin".

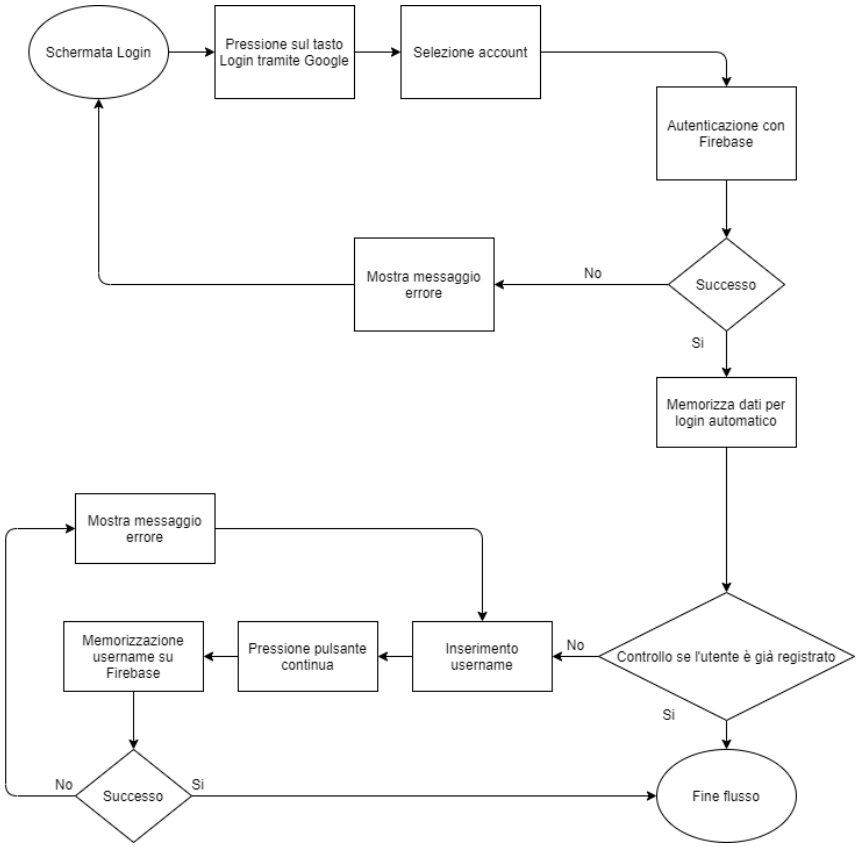

Figura 3.2. Flowchart per il login tramite Google

3.2.3 Mockups

Per la realizzazione dei mockups viene utilizzata una piattaforma gratuita online, denominata InvisionApp, che consente la creazione di mockups in tempo reale a più utenti contemporaneamente. Grazie a tale piattaforma abbiamo realizzato i mockup relativi ad ogni schermata dell'applicazione.

Al primo avvio, dopo aver visualizzato la splashscreen di caricamento, l'utente viene indirizzato alla schermata di login (Figura 3.4) dove può decidere se accedere tramite Google oppure anonimamente.

L'utente che effettua l'accesso con Google, non avendo ancora completato la procedura di registrazione, viene indirizzato ad una videata che richiede l'inserimento dello username (Figura 3.5); non è ammessa la scelta di username già esistenti nel database.

Completato il processo di login/registrazione, l'app reindirizza l'utente sulla home page.

Nella parte superiore di questa (Figura 3.6) vengono riportati l'immagine del profilo, lo username dell'utente e, in successione, un filtro per categoria delle ricette pubblicate. La medesima schermata presenta, nella fascia mediana, un carosello

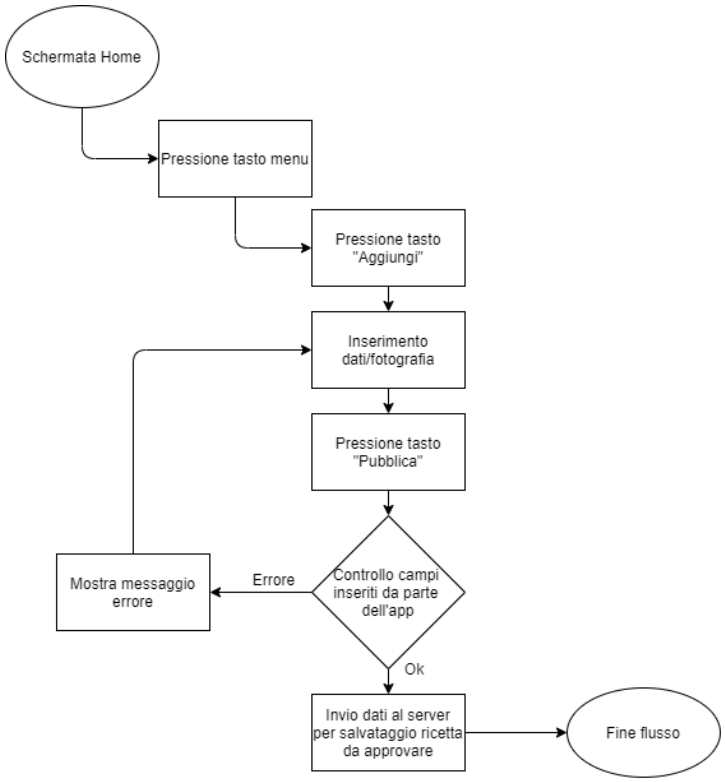

Figura 3.3. Flowchart per l'inserimento di una nuova ricetta

di dieci ricette che può variare in funzione del filtro selezionato mentre, nella parte inferiore, vengono visualizzate le ultime dieci ricette pubblicate e il pulsante "Menù".

Interagendo con il "Menù" l'app mostra altri pulsanti che consentono, dal basso verso l'alto, il collegamento alle seguenti funzionalità:

- Aggiunta di una ricetta.
- Ricerca di una ricetta.
- Gestione delle ricette preferite.
- Impostazioni.

Mentre i pulsanti "Impostazioni" e "Ricerca di una ricetta" producono lo stesso risultato per tutte le tipologie di utente, i pulsanti "Gestione delle ricette preferite" e "Aggiunta di una ricetta" producono output differenti a seconda che l'utente sia, o meno, registrato. In particolare, nel caso di utente anonimo, compare un avviso che indica che la funzionalità è riservata agli utenti registrati e che invita alla registrazione.

La funzionalità "Aggiunta di una ricetta", di cui al mockup di Figura 3.7, consente all'utente registrato di inserire il nome della ricetta, il tempo della sua preparazione, il numero delle porzioni, nonché di indicare gli ingredienti e i passaggi, oltre all'immagine del piatto.

Figura 3.4. Mockup per il login

Figura 3.5. Mockup per la registrazione

Una volta completati tutti i campi richiesti, l'utente registrato preme il tasto di pubblicazione e l'app procede a sottoporre all'autorizzazione dell'"admin" la pubblicazione della ricetta; viceversa, una compilazione non completa di tutti i campi da parte dell'utente comporta la visualizzazione di un messaggio di errore.

La funzionalità "Ricerca di una ricetta", di cui al mockup di Figura 3.8, consente all'utente la ricerca delle ricette contenute nel database semplicemente inserendo,

Figura 3.6. Mockup per la Home Page

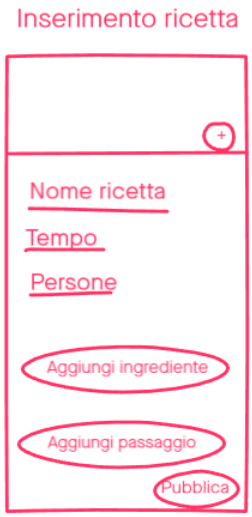

Figura 3.7. Mockup per l'inserimento di una nuova ricetta

anche parzialmente, il nome della ricetta.

La funzionalità "Gestione delle ricette preferite", di cui al mockup di Figura 3.9, mostra l'elenco delle ricette preferite dall'utente registrato.

La funzionalità "Impostazioni", di cui al mockup di Figura 3.10, consente di accedere ai pulsanti di log-out, di variazione dell'immagine del profilo e, nel caso di utente "admin", di mostrare le ricette da approvare.

Nella schermata di approvazione (Figura 3.11), simile a quella della gestione

Figura 3.8. Mockup per la ricerca delle ricette

Figura 3.9. Mockup per la visualizzazione delle ricette preferite

delle ricette preferite, vengono visualizzate tutte le ricette pubblicate ma non ancora revisionate.

L'opzione di visualizzazione di una ricetta (Figura 3.12) consente all'utente registrato (sia "non-admin" che "admin") di cliccare sul pulsante "multiFab" per impostare/rimuovere la ricetta fra i preferiti; nel caso in cui l'utente non sia registrato, tale pulsante viene visualizzato con sfondo grigio e, cliccandolo, mostra un

Figura 3.10. Mockup per le impostazioni

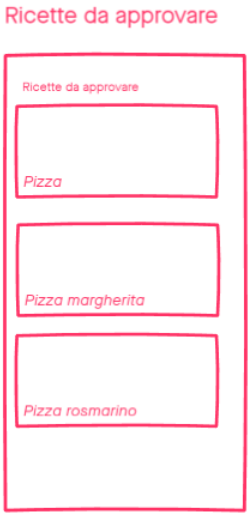

Figura 3.11. Mockup per la visualizzazione delle ricette da approvare

messaggio nel quale si specifica che la funzionalità è riservata agli utenti registrati ed invita alla registrazione.

L'utente "admin", selezionando la ricetta tramite il percorso "Menù" - "Impostazioni" - "Approva ricette", visualizzerà il pulsante "multiFab" con la funzione di approvazione o cancellazione; tale pulsante ha una grafica diversa rispetto a quello indicato in precedenza.

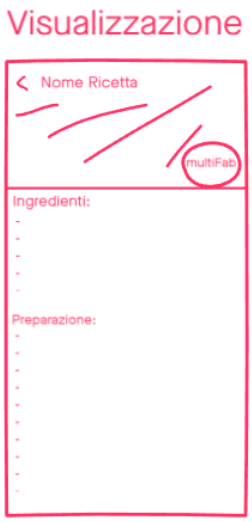

Figura 3.12. Mockup per la visualizzazione di una ricetta

3.2.4 Progettazione della componente dati

Per la gestione dei dati dell'applicazione non si utilizza un database relazionale ma un database NoSQL fornito dalla piattaforma Firebase, realizzata da Google; fra i diversi servizi online forniti da questa piattaforma, *Knife&Spoon* utilizza autenticazione, salvataggio dei dati (database) e memorizzazione (storage) di file.

Firebase, tramite il servizio di database remoto e non relazionale, riunisce documenti in collezioni, identificate da nomi, e li rende accessibili con l'uso del nome del documento stesso; ogni documento contiene elementi che sono identificati da coppie chiave-valore.

Si è scelto di utilizzare la piattaforma Firebase perché, oltre ad offrire uno strumento semplice da usare nella creazione e nella gestione del database, è anche dotata di un notevole grado di sicurezza in senso ampio; l'uso alternativo di un server "personale" SQL comporterebbe anche, fra le altre cose, la necessità di gestire la sicurezza del database ponendolo al riparo da possibili attacchi informatici come, ad esempio, le SQLinjection.

I servizi offerti da Firebase sono, fra l'altro, anche ben documentati e supportati da Android, iOS e pagine web.

Autenticazione

Tramite Firebase Authentication si può gestire facilmente il login e la registrazione degli utenti nell'applicazione.

I dati generati con l'autenticazione tramite Google e con l'autenticazione anonima vengono visualizzati come in Figura 3.13.

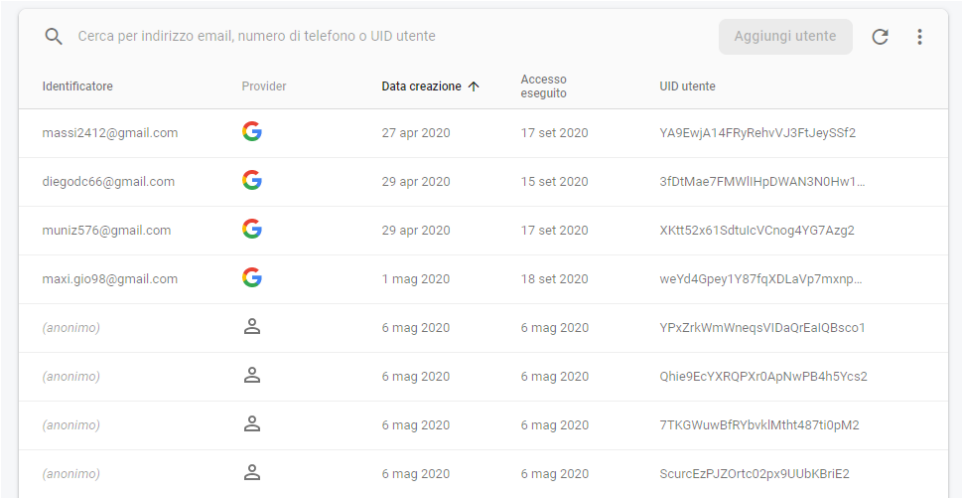

Figura 3.13. Visualizzazione di alcuni dati in Firebase Authentication

Dati permanenti

Si è deciso di dividere il database, offerto nel servizio Firebase Firestore, in due collezioni: "Utenti" e "Ricette".

Per la prima collezione si è deciso di caratterizzare ogni documento da cinque campi:

- *Mail*: di tipo stringa, contiene l'email dell'utente registrato.
- *Nome*: di tipo stringa, contiene lo username scelto dall'utente registrato.
- *Immagine*: di tipo stringa, contiene il link dell'immagine del profilo salvata in Firebase Storage.
- *Preferiti*: di tipo vettore di stringhe, contiene gli id dei documenti delle ricette, presenti nella collezione "Ricette", inserite fra i preferiti dell'utente.
- *isAdmin*: di tipo booleano, indica se l'utente registrato sia un admin, impostando il relativo valore a `true`, o meno.

La Figura 3.14 riporta un esempio di documento di un utente registrato.

Per la collezione "Ricette" si è deciso di caratterizzare i corrispondenti documenti nel seguente modo:

- *Autore*: di tipo stringa, contiene l'id del documento dell'utente, presente nella collezione "Utenti", che ha pubblicato la ricetta.
- *Categoria*: di tipo stringa, contiene le informazioni sulla categoria della ricetta. Questo campo può assumere un valore compreso tra "Antipasto", "Primo", "Secondo", "Contorno" e "Dolce".
- *Numero persone*: di tipo stringa, contiene il numero delle porzioni.
- *Tempo di preparazione*: di tipo stringa, contiene il numero di minuti necessari per preparare la ricetta.
- *Thumbnail*: di tipo stringa, contiene il link dell'immagine della ricetta salvata in Firebase Storage.

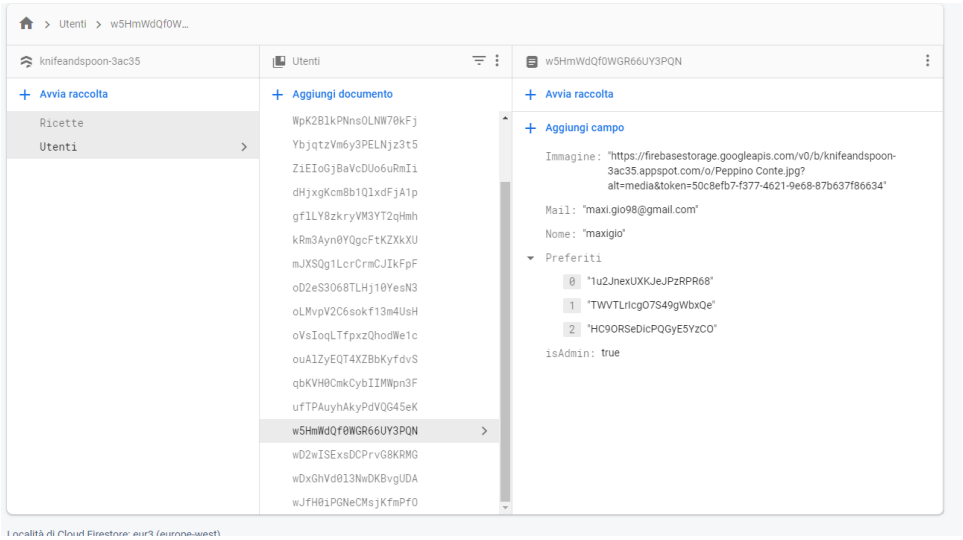

Figura 3.14. Visualizzazione di un documento di un utente

- *Timestamp*: di tipo timestamp, contiene la data, l'ora e il fuso orario nel quale la ricetta è stata pubblicata.
- *Titolo*: di tipo stringa, contiene il nome della ricetta.
- *isApproved*: di tipo booleano, indica se la ricetta è stata approvata (valore `true`) o meno.
- *Ingredienti*: di tipo array di mappe, identificato da un numero sequenziale, riporta i seguenti campi:
  - *Nome*: di tipo stringa, contiene il nome dell'ingrediente.
  - *Quantità*: di tipo stringa, contiene la quantità dell'ingrediente.
  - *Unità di misura*: di tipo stringa, contiene l'unità di misura dell'ingrediente come, ad esempio, "grammi", "litri" etc.
- *Passaggi*: di tipo array di stringhe, contiene i passaggi da effettuare per eseguire la ricetta.

La Figura 3.15 mostra un esempio di un documento di una ricetta.

Storage delle immagini

Nel salvataggio delle immagini, ed al fine di evitare che queste abbiano lo stesso nome, si è deciso di assegnare il nome dei file tramite `UUID` poiché, caricando su Firebase Storage un'immagine con lo stesso nome di un'altra, provocherebbe la sua sovrascrittura.

`UUID`, abbreviazione di Universally Unique Identifier, è un identificativo composto da 16 byte, ossia 128 bit, ed è rappresentato nella sua forma canonica da 32 caratteri esadecimali. È altamente improbabile ottenere una stringa duplicata poiché, con 16^{32} possibili combinazioni, si è calcolato che la probabilità che questo accada sia del 50% se si genera un miliardo di `UUID` al secondo in un periodo di cento anni.

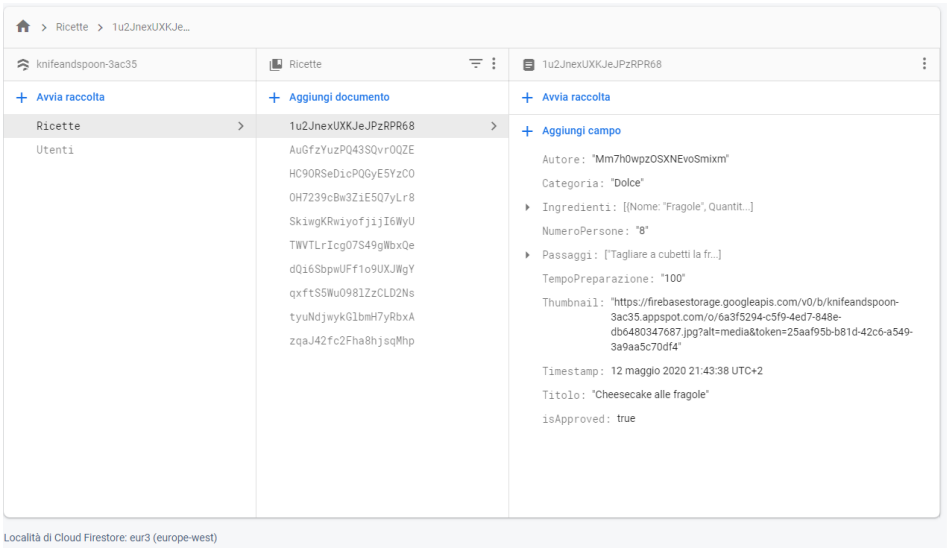

Figura 3.15. Visualizzazione di un documento di una ricetta

Se la ricetta non viene approvata, si effettua l'eliminazione da Firebase Storage dell'immagine ricollegata a quella ricetta.

Di seguito si riporta l'interfaccia utente di Firebase Storage (Figura 3.16).

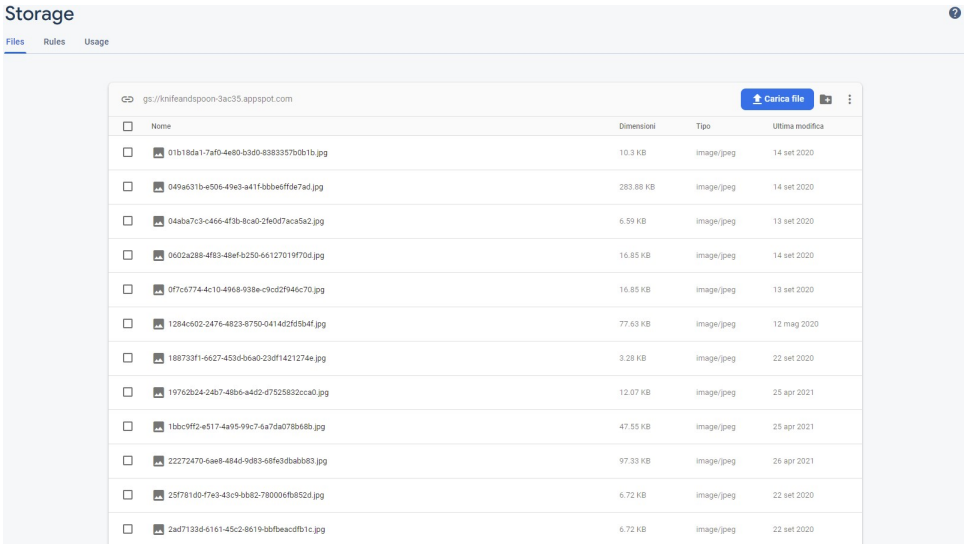

Figura 3.16. Schermata di Firebase Storage

4

Implementazione in Xamarin

In questo capitolo si entrerà nel dettaglio dell'implementazione di Knife&Spoon realizzata tramite Xamarin; in particolare, verranno mostrati gli screenshot dell'applicazione spiegando, nel dettaglio, il funzionamento delle varie componenti ivi presenti.

4.1 Login

Nella Figura 4.1 viene mostrata la schermata dedicata alle funzionalità di registrazione o login.

Figura 4.1. Schermata, realizzata in Xamarin, per effettuare il login o la registrazione

L'utente può scegliere, tramite gli appositi pulsanti, di effettuare l'accesso all'applicazione tramite Google oppure anonimamente.

Una volta selezionata la modalità di accesso, l'applicazione provvede a salvare i dati necessari per effettuare automaticamente i successivi accessi. Una volta ultimato questo passaggio, verrà mostrata la home page (Figura 4.3) oppure, se l'utente ha effettuato il login tramite Google e non ha mai completato la procedura di registrazione, la schermata di inserimento dello username (Figura 4.2).

4.2 Inserimento dello username

Nella Figura 4.2 si riporta la schermata dedicata all'inserimento dello username di un nuovo utente che ha effettuato il login tramite Google.

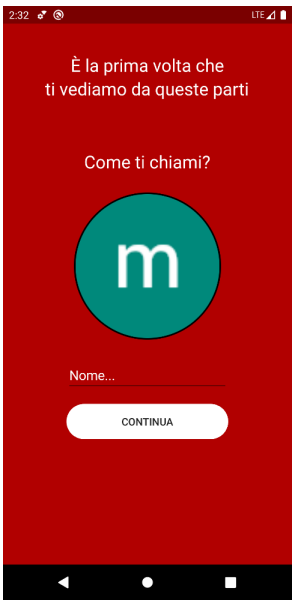

Figura 4.2. Schermata, realizzata in Xamarin, per l'inserimento dello username

L'utente, nell'apposita casella di testo, può scegliere uno username non più lungo di venti caratteri e, inoltre, non può scegliere uno username già presente nel database; scegliendo uno username valido e cliccando sul pulsante "Continua", verrà mostrata all'utente la home page (Figura 4.3).

Se viene inserito uno username troppo corto o già esistente, l'applicazione mostra all'utente un messaggio di errore, tramite il quale lo invita a variare lo username scritto. Se l'utente prova ad inserire oltre venti caratteri, l'applicazione non registrerà nella casella di testo ulteriori nuovi caratteri oltre il ventesimo.

4.3 Home page

Nella Figura 4.3 viene riportata la home page di *Knife&Spoon*; nella parte superiore di questa sono riportati l'immagine del profilo, lo username dell'utente e, in successione, un filtro per categoria delle ricette pubblicate. La medesima schermata presenta, nella fascia mediana, un carosello di dieci ricette che può variare in funzione del filtro selezionato mentre, nella parte inferiore, vengono visualizzate le ultime dieci ricette pubblicate e il pulsante "Menù".

Figura 4.3. Home page realizzata in Xamarin

Quando l'utente sceglie un filtro, oltre a cambiare il contenuto del carosello, viene mostrata un'icona di spunta nel pulsante selezionato. Per disattivare un filtro l'utente può cliccare su un altro filtro, attivandolo, oppure sul filtro attivo, ripristinando, in tal modo, il contenuto del carosello; l'icona di spunta verrà rimossa dal filtro appena disattivato.

Nella parte superiore della home page l'utente può effettuare uno swipe dall'alto verso il basso e l'app ricaricherà il contenuto della home page con un'animazione specifica, ripristinando, anche, l'eventuale selezione di un filtro.

Interagendo con il "Menù" l'app mostra altri pulsanti che consentono, dal basso verso l'alto, il collegamento alle seguenti funzionalità:

- aggiunta di una ricetta;
- ricerca di una ricetta;
- gestione delle ricette preferite;
- impostazioni.

Mentre i pulsanti “Impostazioni” e “Cerca” producono lo stesso risultato per tutte le tipologie di utente, i pulsanti “Preferiti” e “Aggiunta” producono output differenti a seconda che l’utente sia, o meno, registrato. In particolare, nel caso di utente anonimo, compare un avviso che indica che la funzionalità è riservata agli utenti registrati e che invita alla registrazione.

Inoltre, se l’utente seleziona una ricetta presente nel carosello o nell’elenco delle ultime dieci ricette pubblicate, viene reindirizzato alla schermata di visualizzazione della ricetta selezionata (Figura 4.7).

4.4 Aggiunta di una ricetta

Nella Figura 4.4 viene riportata la schermata relativa all’inserimento di una nuova ricetta; nella parte superiore è presente un pulsante con l’icona di una fotocamera che, se cliccato, consente all’utente di inserire, nell’apposito spazio, un’immagine presente nella galleria delle immagini o scattata al momento tramite la fotocamera.

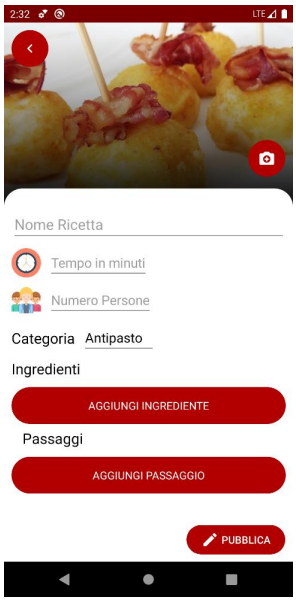

Figura 4.4. Schermata, realizzata in Xamarin, per scrivere una nuova ricetta

L’utente deve compilare sia il campo “Nome Ricetta”, inserendo il titolo prescelto, sia il campo “Tempo in minuti”, indicando il tempo di preparazione espresso in minuti, sia, infine, il campo “Numero Persone”, specificando il numero delle porzioni; egli, infine, dovrà scegliere la categoria (antipasto, primo, etc.) cui la ricetta appartiene.

L’utente, per poter aggiungere un ingrediente alla ricetta, deve cliccare sul pulsante “Aggiungi ingrediente”; vengono, così, mostrate due caselle di testo che consentono, rispettivamente, di specificare il nome e la quantità dell’ingrediente. È,

inoltre, possibile sia scegliere l'unità di misura della quantità, con apposito selettore, che rimuovere l'ingrediente con il pulsante specifico posto a sinistra dello stesso.

Selezionando l'unità di misura "q.b." verrà disabilitato il campo relativo alla quantità dell'ingrediente coinvolto.

Toccando sul pulsante "Aggiungi passaggio", l'utente deve inserire almeno una descrizione delle modalità di preparazione della ricetta; anche in questo caso è possibile rimuovere il passaggio con il pulsante specifico posto a sinistra dello stesso.

Una volta riempiti tutti i campi e inserita un'immagine del piatto, l'utente può caricare la ricetta cliccando sul pulsante "Pubblica", posizionato nella parte inferiore destra della schermata; una volta attivato tale pulsante, viene avviata la procedura di upload dei dati relativi alla ricetta, mostrando una schermata di caricamento. Terminato tale processo, l'utente viene rimandato alla home page (Figura 4.3).

Se l'utente non ha compilato tutti i campi obbligatori e clicca sul pulsante "Pubblica", l'applicazione mostra un messaggio di errore che lo invita a completare tutte le informazioni richieste.

4.5 Ricerca di una ricetta

Nella Figura 4.5 si riporta la schermata relativa alla ricerca delle ricette. L'utente, cliccando sulla casella di testo posizionata nella parte superiore dello schermo, può inserire un titolo di una ricetta, anche parziale. Cliccando, poi, sulla lente di ingrandimento posizionata alla sua destra, o premendo il pulsante "Invio" della tastiera, l'applicazione visualizzerà le ricette trovate nel database sotto forma di elenco.

Figura 4.5. Schermata, realizzata in Xamarin, per la ricerca delle ricette

Selezionando uno dei risultati proposti, l'app mostra la schermata di visualizzazione della ricetta corrispondente (Figura 4.7).

Nel caso in cui non siano presenti nel database ricette con il titolo specificato dall'utente, l'app mostrerà un messaggio di esito negativo.

4.6 Gestione delle ricette preferite

Nella Figura 4.6 si riporta la schermata relativa alla visualizzazione delle ricette preferite. Tale schermata riporta l'elenco delle ricette che l'utente ha precedentemente impostato come preferite; si potrà visualizzare una ricetta semplicemente selezionandola dall'elenco stesso.

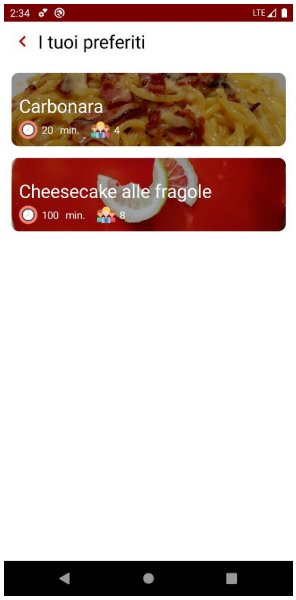

Figura 4.6. Schermata, realizzata in Xamarin, per la visualizzazione delle ricette preferite

L'utente può rimuovere una ricetta dai preferiti; qualora ciò accada, la ricetta non viene più visualizzata nell'elenco sopra descritto. Un messaggio di assenza delle ricette preferite viene visualizzato dall'app se non sono state impostate delle ricette come preferite o se l'utente ha rimosso l'unica ricetta preferita.

4.7 Visualizzazione di una ricetta approvata

Una volta che l'amministratore ha approvato una ricetta, questa risulta visualizzabile da tutti gli utenti di *Knife&Spoon*, come rilevabile dalla schermata di Figura 4.7.

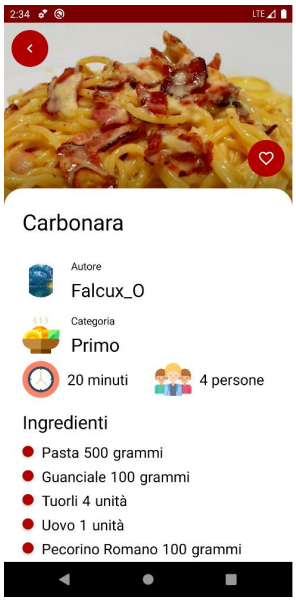

Figura 4.7. Schermata, realizzata in Xamarin, per la visualizzazione di una ricetta approvata

Nella parte superiore di tale schermata sono presenti sia un pulsante che consente di tornare alla schermata precedente, sia l'immagine del piatto, sia il pulsante di aggiunta/rimozione della ricetta dai preferiti (icona di un cuore). Gli utenti registrati possono selezionare il pulsante con l'icona di un cuore che apparirà vuoto, nel caso in cui la ricetta non rientra fra quelle preferite dall'utente stesso, o pieno, nel caso opposto. Dal punto di vista del database, la selezione del pulsante con il cuore comporta il salvataggio/rimozione, dal vettore *Preferiti*, presente nel documento dell'utente attuale, dell'id del documento relativo alla ricetta visualizzata. Nel caso di utente anonimo, cliccando sul pulsante del cuore, l'applicazione mostrerà un messaggio di invito alla registrazione.

Nella parte inferiore della schermata sono riportati il nome e l'immagine del profilo dell'autore della ricetta, la categoria con la relativa icona, il tempo di preparazione, il numero delle porzioni e, infine, gli ingredienti e i passaggi.

4.8 Impostazioni

Nella Figura 4.8 viene riportata la schermata relativa alle impostazioni; i pulsanti a disposizione dell'utente varieranno a seconda che lo stesso sia anonimo, registrato o amministratore.

Se l'utente è anonimo, ha a disposizione solo il pulsante "Effetta il log-out" che lo riporta alla schermata di registrazione (Figura 4.1); l'applicazione in background provvederà alla rimozione dei dati salvati necessari per l'autenticazione automatica.

Se l'utente è registrato, oltre a poter cliccare sul pulsante precedentemente riportato, avrà a disposizione anche il pulsante "Cambia immagine profilo". Selezionando

Figura 4.8. Schermata, realizzata in Xamarin, per la visualizzazione delle impostazioni realizzata in Xamarin

tale pulsante, l'utente può variare la fotografia del suo profilo usando un'immagine proveniente dalla fotocamera oppure una proveniente dalla galleria.

Se l'utente è amministratore, oltre alle funzionalità precedenti, ne possiede una terza accessibile dal pulsante "Approva ricette". Quando tale pulsante viene cliccato, si aprirà la schermata contenente l'elenco delle ricette da approvare (Figura 4.9).

4.9 Visualizzazione delle ricette da approvare

Nella Figura 4.9 viene mostrata la schermata relativa alla visualizzazione delle ricette non revisionate.

Se esistono delle ricette da approvare, queste verranno mostrate tramite un elenco; in caso contrario all'amministratore verrà mostrato un avviso che lo informerà sul fatto che non esistono delle ricette da revisionare.

4.10 Visualizzazione di una ricetta da approvare

Nella Figura 4.10 viene riportata la schermata relativa alla visualizzazione di una ricetta da approvare; tale videata è simile a quella riportata nella Figura 4.7, relativa a una ricetta approvata. L'unica differenza rilevabile è la sostituzione del pulsante di aggiunta/rimozione della ricetta ai preferiti con il pulsante di approvazione/rimozione della ricetta.

Figura 4.9. Schermata, realizzata in Xamarin, per la visualizzazione delle ricette da approvare

Figura 4.10. Schermata, realizzata in Xamarin, per la visualizzazione di una ricetta da approvare

Con l'approvazione della ricetta l'applicazione imposterà, nel documento contenuto nel database e relativo ad esso, il valore `true` sulla variabile `isApproved` e ne consentirà la visualizzazione a tutti gli utenti.

Con la rimozione della ricetta, l'applicazione provvederà all'eliminazione dal database della ricetta selezionata e della relativa immagine.

5

Implementazione in Flutter

In questo capitolo si entrerà nel dettaglio dell'implementazione di Knife&Spoon realizzata tramite Flutter; in particolare, verranno mostrati gli screenshot dell'applicazione spiegando, nel dettaglio, il funzionamento delle varie componenti ivi presenti.

5.1 Login

Nella Figura 5.1 viene mostrata la schermata dedicata alle funzionalità di registrazione o login.

Figura 5.1. Schermata, realizzata in Flutter, per effettuare il login o la registrazione

L'utente può scegliere, tramite gli appositi pulsanti, di effettuare l'accesso all'applicazione tramite Google oppure anonimamente.

Una volta selezionata la modalità di accesso, l'applicazione provvede a salvare i dati necessari per effettuare automaticamente i successivi accessi. Una volta ultimato questo passaggio, verrà mostrata la home page (Figura 5.3) oppure, se l'utente ha effettuato il login tramite Google e non ha mai completato la procedura di registrazione, la schermata di inserimento dello username (Figura 5.2).

5.2 Inserimento dello username

Nella Figura 5.2 si riporta la schermata dedicata all'inserimento dello username di un nuovo utente che ha effettuato il login tramite Google.

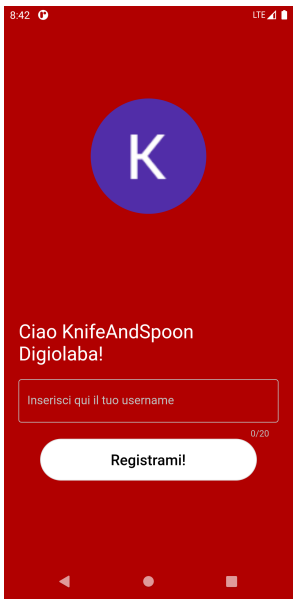

Figura 5.2. Schermata, realizzata in Flutter, per l'inserimento dello username

L'utente, nell'apposita casella di testo, può scegliere uno username non più lungo di venti caratteri e, inoltre, non può scegliere uno username già presente nel database; scegliendo uno username valido e cliccando sul pulsante "Continua", verrà mostrata all'utente la home page (Figura 5.3).

Se viene inserito uno username troppo corto o uno già esistente, l'applicazione mostra all'utente un messaggio di errore, tramite il quale lo invita a variare lo username inserito. Se l'utente prova ad inserire oltre venti caratteri, l'applicazione non registrerà nella casella di testo ulteriori nuovi caratteri oltre il ventesimo.

5.3 Home page

Nella Figura 5.3 viene riportata la home page, realizzata in Flutter, di *Knife&Spoon*; nella parte superiore di questa sono riportati l'immagine del profilo, lo username dell'utente e, in successione, un filtro per ogni categoria delle ricette pubblicate. La medesima schermata presenta, nella fascia mediana, un carosello di dieci ricette che può variare in funzione del filtro selezionato, mentre, nella parte inferiore, vengono visualizzate le ultime dieci ricette pubblicate e il pulsante "Menù".

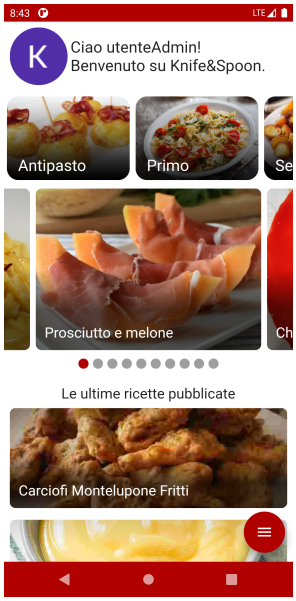

Figura 5.3. Home page realizzata in Flutter

Quando l'utente sceglie un filtro, oltre a cambiare il contenuto del carosello, viene mostrata un'icona di spunta nel pulsante selezionato. Per disattivare un filtro l'utente può cliccare su un altro filtro, attivando quest'ultimo, oppure può cliccare sul filtro attivo, ripristinando, in tal modo, il contenuto del carosello; l'icona di spunta verrà rimossa dal filtro appena disattivato.

Nella parte superiore della home page l'utente può effettuare uno swipe dall'alto verso il basso; a seguito di ciò l'app ricaricherà il contenuto della home page con un'animazione specifica, ripristinando, anche, l'eventuale selezione di un filtro.

Interagendo con il "Menù" l'app mostra altri pulsanti che consentono, dal basso verso l'alto, il collegamento alle seguenti funzionalità:

- aggiunta di una ricetta;
- ricerca di una ricetta;
- gestione delle ricette preferite;
- impostazioni.

Mentre i pulsanti “Impostazioni” e “Cerca” producono lo stesso risultato per tutte le tipologie di utente, i pulsanti “Preferiti” e “Aggiunta” producono output differenti a seconda che l’utente sia, o meno, registrato. In particolare, nel caso di utente anonimo, compare un avviso che indica che la funzionalità è riservata agli utenti registrati invitando alla registrazione.

Inoltre, se l’utente seleziona una ricetta presente nel carosello o nell’elenco delle ultime dieci ricette pubblicate, egli viene reindirizzato alla schermata di visualizzazione della ricetta selezionata (Figura 5.7).

5.4 Aggiunta di una ricetta

Nella Figura 5.4 viene riportata la schermata relativa all’inserimento di una nuova ricetta; nella parte superiore è presente un pulsante con l’icona di una fotocamera che, se cliccato, consente all’utente di inserire, nell’apposito spazio, un’immagine presente nella galleria delle immagini o scattata al momento tramite la fotocamera.

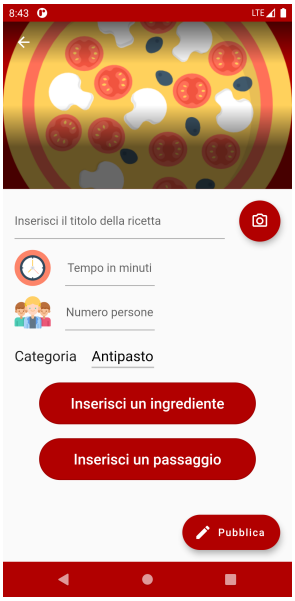

Figura 5.4. Schermata, realizzata in Flutter, per scrivere una nuova ricetta

L’utente deve compilare sia il campo “Inserisci il titolo della ricetta”, inserendo il titolo prescelto, sia il campo “Tempo in minuti”, indicando il tempo di preparazione espresso in minuti, sia, infine, il campo “Numero persone”, specificando il numero delle porzioni; egli, infine, dovrà scegliere la categoria (antipasto, primo, etc.) cui la ricetta appartiene.

L’utente, per poter aggiungere un ingrediente alla ricetta, deve cliccare sul pulsante “Inserisci un ingrediente”; vengono, così, mostrate due caselle di testo che consentono, rispettivamente, di specificare il nome e la quantità dell’ingrediente. È,

inoltre, possibile sia scegliere l'unità di misura della quantità, con apposito selettore, che rimuovere l'ingrediente, tramite il pulsante specifico posto a sinistra dello stesso.

Selezionando l'unità di misura "q.b." verrà disabilitato il campo relativo alla quantità dell'ingrediente coinvolto.

Toccando sul pulsante "Inserisci un passaggio", l'utente deve inserire almeno una descrizione delle modalità di preparazione della ricetta; anche in questo caso è possibile rimuovere il passaggio tramite il pulsante specifico posto a sinistra dello stesso.

Una volta riempiti tutti i campi e inserita un'immagine del piatto, l'utente può caricare la ricetta cliccando sul pulsante "Pubblica", posizionato nella parte inferiore destra della schermata; una volta attivato tale pulsante, viene avviata la procedura di upload dei dati relativi alla ricetta, mostrando una schermata di caricamento. Terminato tale processo, l'utente viene rimandato alla home page (Figura 5.3).

Se l'utente non ha compilato tutti i campi obbligatori e clicca sul pulsante "Pubblica", l'applicazione mostra un messaggio di errore che lo invita a completare tutte le informazioni richieste.

5.5 Ricerca di una ricetta

Nella Figura 5.5 si riporta la schermata relativa alla ricerca delle ricette. L'utente, cliccando sul pulsante della lente di ingrandimento, potrà inserire sulla casella di testo che apparirà, un titolo di una ricetta, anche parziale. Se l'utente volesse eliminare il titolo scritto, può cliccare sul pulsante con l'icona della croce, posizionato alla destra della casella di testo; altrimenti, premendo il pulsante "Invio" della tastiera, l'applicazione visualizzerà le ricette trovate nel database sotto forma di elenco.

Selezionando uno dei risultati proposti, l'app mostra la schermata di visualizzazione della ricetta corrispondente (Figura 5.7).

Nel caso in cui non siano presenti nel database ricette con il titolo specificato dall'utente, l'app visualizzerà un messaggio di esito negativo.

5.6 Gestione delle ricette preferite

Nella Figura 5.6 viene riportata la schermata relativa alla visualizzazione delle ricette preferite. Tale schermata mostra l'elenco delle ricette che l'utente ha precedentemente impostato come preferite; si potrà visualizzare una ricetta semplicemente selezionandola dall'elenco stesso.

L'utente può rimuovere una ricetta dai preferiti; qualora ciò accada, la ricetta non viene più visualizzata nell'elenco sopra descritto. Un messaggio di assenza delle ricette preferite viene visualizzato dall'app se non sono state impostate delle ricette come preferite, o se l'utente ha rimosso l'unica ricetta preferita.

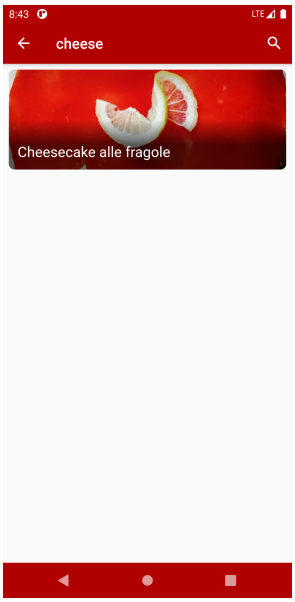

Figura 5.5. Schermata, realizzata in Flutter, per la ricerca delle ricette

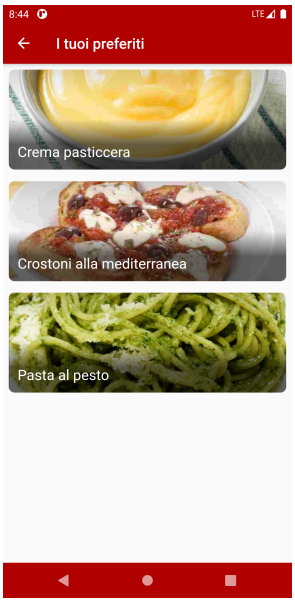

Figura 5.6. Schermata, realizzata in Flutter, per la visualizzazione delle ricette preferite

5.7 Visualizzazione di una ricetta approvata

Una volta che l'amministratore ha approvato una ricetta, questa risulta visualizzabile da tutti gli utenti di *Knife&Spoon*, come rilevabile dalla schermata di Figura

5.7.

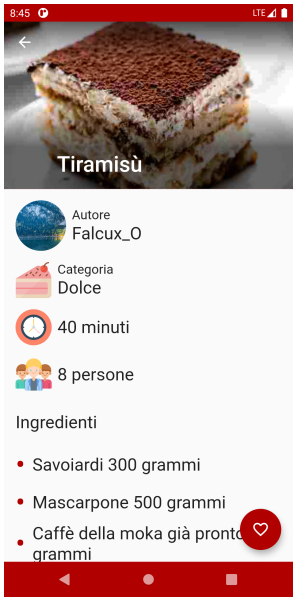

Figura 5.7. Schermata, realizzata in Flutter, per la visualizzazione di una ricetta approvata

Nella parte superiore di tale schermata sono presenti sia un pulsante che consente di tornare alla schermata precedente, sia l'immagine del piatto, sia il pulsante di aggiunta/rimozione della ricetta dai preferiti (icona di un cuore). Gli utenti registrati possono selezionare il pulsante con l'icona di un cuore che apparirà vuoto, nel caso in cui la ricetta non rientra fra quelle preferite dall'utente stesso, o pieno, nel caso opposto. Dal punto di vista del database, la selezione del pulsante con il cuore comporta il salvataggio/la rimozione, dal vettore *Preferiti*, presente nel documento dell'utente attuale, dell'id del documento relativo alla ricetta visualizzata. Nel caso di utente anonimo, cliccando sul pulsante del cuore, l'applicazione mostrerà un messaggio di invito alla registrazione.

Nella parte inferiore della schermata sono riportati il nome e l'immagine del profilo dell'autore della ricetta, la categoria con la relativa icona, il tempo di preparazione, il numero delle porzioni e, infine, gli ingredienti e i passaggi.

5.8 Impostazioni

Nella Figura 5.8 viene riportata la schermata relativa alle impostazioni; i pulsanti a disposizione dell'utente varieranno a seconda che lo stesso sia anonimo, registrato o amministratore.

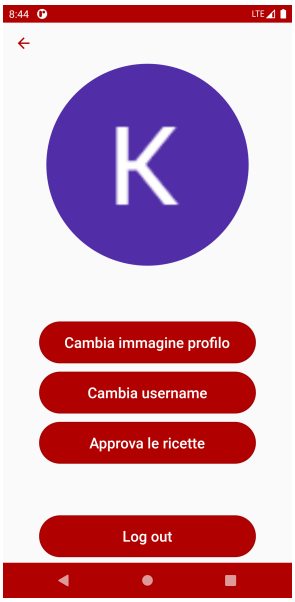

Figura 5.8. Schermata, realizzata in Flutter, per la visualizzazione delle impostazioni

Se l'utente è anonimo, egli ha a disposizione solo il pulsante "Log out", che lo riporta alla schermata di registrazione (Figura 5.1); l'applicazione in background provvederà alla rimozione dei dati salvati necessari per l'autenticazione automatica.

Se l'utente è registrato, oltre a poter cliccare sul pulsante precedentemente riportato, avrà a disposizione anche i pulsanti "Cambia immagine profilo" e "Cambia username". Selezionando il primo, l'utente può variare la fotografia del suo profilo usando un'immagine proveniente dalla fotocamera, oppure una proveniente dalla galleria. Selezionando, invece, il secondo, l'utente può cambiare il proprio username con uno che non sia presente nel database e che abbia una lunghezza tra i cinque e i venti caratteri.

Se l'utente è amministratore, oltre alle funzionalità precedenti, ne possiede una quarta accessibile dal pulsante "Approva le ricette". Quando tale pulsante viene cliccato, si aprirà la schermata contenente l'elenco delle ricette da approvare (Figura 5.9).

5.9 Visualizzazione delle ricette da approvare

Nella Figura 5.9 viene mostrata la schermata relativa alla visualizzazione delle ricette non revisionate.

Se esistono delle ricette da approvare, queste verranno mostrate tramite un elenco; in caso contrario all'amministratore verrà mostrato un avviso che lo informerà sul fatto che non esistono delle ricette da revisionare.

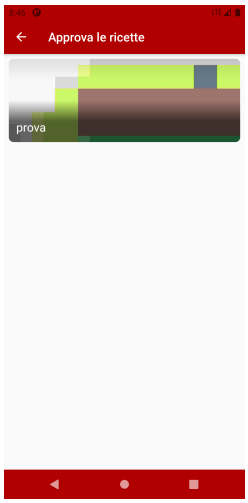

Figura 5.9. Schermata, realizzata in Flutter, per la visualizzazione delle ricette da approvare

5.10 Visualizzazione di una ricetta da approvare

Nella Figura 5.10 viene riportata la schermata relativa alla visualizzazione di una ricetta da approvare; tale videata è simile a quella riportata nella Figura 5.7, relativa a una ricetta approvata. L'unica differenza rilevabile è la sostituzione del pulsante di aggiunta/rimozione della ricetta ai preferiti con il pulsante di approvazione/rimozione della ricetta.

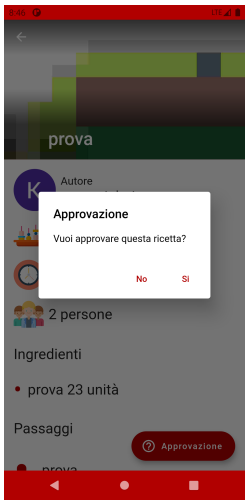

Figura 5.10. Schermata, realizzata in Flutter, per la visualizzazione di una ricetta da approvare

Con l'approvazione della ricetta l'applicazione imposterà, nel documento contenuto nel database e relativo ad esso, il valore `true` sulla variabile `isApproved`, e ne consentirà la visualizzazione a tutti gli utenti.

Con la rimozione della ricetta, l'applicazione provvederà all'eliminazione, dal database, della ricetta selezionata e della relativa immagine.

6

Confronto fra le due implementazioni

In questo capitolo si confronteranno alcuni aspetti delle implementazioni di Knife&Spoon in Xamarin e in Flutter; in particolare, verranno mostrate alcune delle differenze nell'interfaccia grafica, nelle modalità di interazione con i vari servizi di Firebase e, infine, nella dimensione dei file APK.

6.1 Interfaccia grafica

L'interfaccia grafica, nelle applicazioni create tramite Xamarin e Flutter, risulta intuitiva e fluida, consentendo un'esperienza d'uso piacevole per l'utente.

Si è cercato di mantenere lo stesso stile grafico in entrambe le implementazioni anche se, in alcune schermate, sono presenti delle differenze che non cambiano significativamente l'esperienza d'uso dell'utente.

La differenza più significativa si trova nella home page (Figure 4.3 e 5.3), dove il pulsante che svolge la funzione di menù ha un aspetto diverso a seconda dell'implementazione. Più specificatamente:

- Nell'implementazione in Xamarin il pulsante di menù ha un'icona di una pizza che, se premuto, ruota in senso antiorario e mostra altri quattro pulsanti che consentono, rispettivamente, l'accesso alle funzionalità di inserimento, ricerca, gestione delle ricette preferite e, infine, impostazioni (Figura 6.1).
- Nell'implementazione in Flutter il pulsante di menù ha un'icona con tre linee orizzontali che, una volta premuto, cambia, con un'animazione, la propria icona; viene oscurato il resto della schermata e, inoltre, vengono mostrati quattro pulsanti che consentono, rispettivamente, l'accesso alle funzionalità di inserimento, ricerca, gestione delle ricette preferite e, infine, impostazioni (Figura 6.2).

6.2 Interazione con Firebase

Knife&Spoon interagisce numerose volte con i servizi forniti da Firebase, consentendo l'autenticazione degli utenti, la gestione dei loro dati e quelli delle ricette e,

Figura 6.1. Home page, realizzata in Xamarin, con il menù aperto

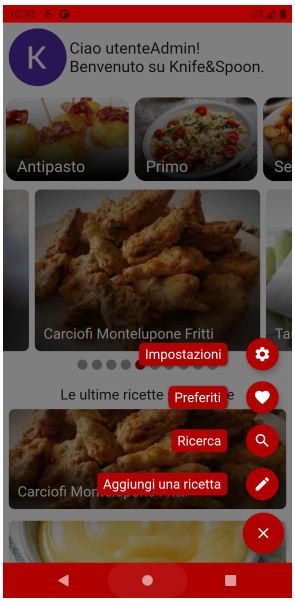

Figura 6.2. Home page, realizzata in Flutter, con il menù aperto

infine, la gestione delle immagini relative ai profili dei vari utenti e dei piatti delle ricette.

Per poter svolgere le operazioni sopra elencate sono state usate alcune tecniche che verranno approfondite nei passi successivi.

6.2.1 Autenticazione

Xamarin

Nell'implementazione in Xamarin, per poter effettuare l'autenticazione dell'utente, si sono usati i plugin `GoogleClientPlugin` (`https://github.com/CrossGeeks/GoogleClientPlugin`), che consente la gestione dell'autenticazione tramite Google, e `Plugin.FirebaseAuth` (`https://github.com/f-miyu/Plugin.FirebaseAuth`), che consente l'interazione con il servizio di autenticazione di Firebase. Entrambi i plugin cross-platfrom utilizzati sono stati sviluppati da terzi poiché, per Xamarin, non esistono plugin cross-platform supportati ufficialmente che svolgono le funzioni descritte in precedenza.

Se l'utente, nella schermata di login/registrazione (Figura 4.1), clicca il pulsante "Sign in with Google", l'applicazione eseguirà il codice che può essere visualizzato, in maniera parziale, nel Listato 6.1.

```
...

private readonly IGoogleClientManager _googleClientManager;

public MainPage()
        {
            _googleClientManager = CrossGoogleClient.Current;
            InitializeComponent();
            CheckUser();
        }

...

private async void LoginAsync()
        {
            anonimous.IsEnabled = false;
            google.IsEnabled = false;
            _googleClientManager.OnLogin += OnLoginCompleted;
            try
            {
                await _googleClientManager.LoginAsync();
            }
            catch (GoogleClientSignInNetworkErrorException e)
            {
                await Navigation.PushModalAsync(new ErrorDialog(e.Message));
                anonimous.IsEnabled = true;
                google.IsEnabled = true;
            }
            catch (GoogleClientSignInCanceledErrorException e)
            {
                await Navigation.PushModalAsync(new ErrorDialog(e.Message));
                anonimous.IsEnabled = true;
                google.IsEnabled = true;
            }
            catch (GoogleClientSignInInvalidAccountErrorException e)
            {
                await Navigation.PushModalAsync(new ErrorDialog(e.Message));
                anonimous.IsEnabled = true;
                google.IsEnabled = true;
            }
            catch (GoogleClientSignInInternalErrorException e)
            {
                await Navigation.PushModalAsync(new ErrorDialog(e.Message));
                anonimous.IsEnabled = true;
                google.IsEnabled = true;
            }
            catch (GoogleClientNotInitializedErrorException e)
            {
                await Navigation.PushModalAsync(new ErrorDialog(e.Message));
                anonimous.IsEnabled = true;
                google.IsEnabled = true;
            }
            catch (GoogleClientBaseException e)
            {
                await Navigation.PushModalAsync(new ErrorDialog(e.Message));
                anonimous.IsEnabled = true;
                google.IsEnabled = true;
            }

        }

...
```

```
        private void OnLoginCompleted(object sender, GoogleClientResultEventArgs<GoogleUser> loginEventArgs)
        {
            if (loginEventArgs.Data != null)
            {
                GoogleUser googleUser = loginEventArgs.Data;
                finalizeLogin(CrossGoogleClient.Current.IdToken, CrossGoogleClient.Current.AccessToken);
            }
            else
            {
                Navigation.PushModalAsync(new ErrorDialog(loginEventArgs.Message));
            }

            _googleClientManager.OnLogin -= OnLoginCompleted;

        }

        private async void finalizeLogin(string idToken, string accessToken)
        {
            var credential = CrossFirebaseAuth.Current.GoogleAuthProvider.GetCredential(idToken, accessToken);
            var result = await CrossFirebaseAuth.Current.Instance.SignInWithCredentialAsync(credential);
            var glob = await CrossCloudFirestore.Current.Instance.GetCollection("Utenti").WhereEqualsTo("Mail",
                CrossFirebaseAuth.Current.Instance.CurrentUser.Email).GetDocumentsAsync();
            if (glob.Count == 0)
            {
                PushPage(new RegisterPage());
            }
            else
            {
                App.Current.MainPage = new NavigationPage(new HomePage());
            }

        }
...
```

Listato 6.1. Porzione di codice per l'autenticazione tramite Google, presente nel file `MainPage.xaml.cs`

L'applicazione, tramite l'istruzione `_googleClientManager.OnLogin + = OnLoginCompleted`, avvia il client manager di Google e, con l'istruzione successiva, prova ad effettuare il login; se non si sono verificati errori, verrà eseguita l'istruzione `finalizeLogin( CrossGoogleClient.Current.IdToken, CrossGoogleClient. Current.AccessToken)`.

I parametri riportati all'interno della chiamata a funzione, ottenuti con l'uso del plugin `GoogleClientPlugin`, contengono i dati necessari per poter effettuare un'autenticazione sicura su Firebase Authentication, tramite il protocollo *OAuth2*.

Nel metodo `finalizeLogin` viene effettuata l'autenticazione su Firebase Authentication mediante l'istruzione `await CrossFirebaseAuth.Current.Instance. SignInWithCredentialAsync(credential)`; tramite l'istruzione successiva, l'applicazione controllerà se l'utente è già registrato.

Se l'utente, nella schermata di login/registrazione (Figura 4.3), clicca nel pulsante "Entra senza account", l'applicazione eseguirà il codice riportato nel Listato 6.2.

```
...

private async void LoginAnonimousAsync()
        {
            loadOverlay.IsVisible = true;
            await CrossFirebaseAuth.Current.Instance.SignInAnonymouslyAsync();
            loadOverlay.IsVisible = false;
            anonimous.IsEnabled = true;
            google.IsEnabled = true;
            App.Current.MainPage = new NavigationPage(new HomePage());
        }

...
```

Listato 6.2. Metodo `LoginAnonimousAsync` presente nel file `MainPage.xaml.cs`

Tramite l'istruzione `await CrossFirebaseAuth.Current.Instance.SignInAnonymouslyAsync()` l'applicazione provvede all'autenticazione anonima su Firebase Authentication e, successivamente, mostra all'utente la home page.

Nel Listato 6.3 viene riportato una porzione di codice, presente nel file `SettingsPage.xaml.cs`, che consente di effettuare l'operazione di log out sia all'utente registrato che a quello anonimo.

```
...

private void LogOut(object sender, EventArgs args)
       {
           CrossFirebaseAuth.Current.Instance.SignOut();
           CrossGoogleClient.Current.Logout();
           App.Current.MainPage = new NavigationPage(new MainPage());
       }

...
```

Listato 6.3. Metodo `LogOut` presente nel file `SettingsPage.xaml.cs`

Flutter

Nell'implementazione in Flutter, per poter effettuare l'autenticazione dell'utente, vengono utilizzati i pacchetti `google_sign_in` (`https://pub.dev/packages/google_sign_in`), che consente la gestione dell'autenticazione dell'utente tramite Google, e `firebase_auth` (`https://pub.dev/packages/firebase_auth`), che consente l'interazione con il servizio di autenticazione di Firebase; entrambi i pacchetti sono sviluppati e supportati ufficialmente, rispettivamente, dagli sviluppatori di Flutter e di Firebase.

Nella schermata di login/registrazione (Figura 5.1), se l'utente clicca sul pulsante "Sign in with Google", l'applicazione eseguirà il codice che, in maniera parziale, è riportato nel Listato 6.4.

```
...

static Future<User> signInWithGoogle({@required BuildContext context}) async {
    FirebaseAuth auth = FirebaseAuth.instance;
    User user;
    final GoogleSignIn googleSignIn = GoogleSignIn();
    final GoogleSignInAccount googleSignInAccount = await googleSignIn.signIn();
    if (googleSignInAccount != null) {
      final GoogleSignInAuthentication googleSignInAuthentication =
          await googleSignInAccount.authentication;

      final AuthCredential credential = GoogleAuthProvider.credential(
        accessToken: googleSignInAuthentication.accessToken,
        idToken: googleSignInAuthentication.idToken,
      );
      try {
        final UserCredential userCredential =
            await auth.signInWithCredential(credential);

        user = userCredential.user;
      }

     ...

      return user;
    }
  }

...
```

Listato 6.4. Porzione di codice per l'autenticazione tramite Google, presente nel file `authentication.dart`

Nel listato sopra riportato è presente l'istruzione `final GoogleSignInAccount googleSignInAccount = await googleSignIn.signIn()`, che provvede ad avviare il client di Google per il login.

Se la procedura si conclude senza errori, tramite le istruzioni successive, vengono salvati nella variabile `credential` i dati necessari per effettuare un'autenticazione sicura su Firebase Authentication con il protocollo *OAuth2* e, tramite l'istruzione `await auth.signInWithCredential(credential)`, avviene l'autenticazione sul servizio Firebase.

Nella schermata di login/registrazione (Figura 5.1), se l'utente clicca sul pulsante "Entra anonimamente", l'applicazione eseguirà il codice che, in maniera parziale, è riportato nel Listato 6.5.

```
...

 static Future<User> signInAnonymously() async {
  User user;
  FirebaseAuth auth = FirebaseAuth.instance;
  await auth.signInAnonymously().then((value) {
    user = value.user;
  });
  return user;
 }

...
```

Listato 6.5. Metodo `signInAnonymously` presente nel file `authentication.dart`

Il metodo riportato nel listato precedente consente di effettuare l'autenticazione di un utente anonimo su Firebase Authentication.

Quando l'utente clicca sul pulsante di log out, l'applicazione eseguirà il codice riportato, parzialmente, nel Listato 6.6.

```
...

  static Future<void> signOut({@required BuildContext context}) async {
   final GoogleSignIn googleSignIn = GoogleSignIn();

   try {
     if (!kIsWeb) {
       await googleSignIn.signOut();
     }
     await FirebaseAuth.instance.signOut();
   } catch (e) {
     ScaffoldMessenger.of(context).showSnackBar(
       Authentication.customSnackBar(
         content: 'Error signing out. Try again.',
       ),
     );
   }
  }

...
```

Listato 6.6. Metodo `signOut` presente nel file `authentication.dart`

6.2.2 Database

Xamarin

Nell'implementazione in Xamarin, per poter effettuare interazioni con il database online, si è usato il plugin `Plugin.CloudFirestore` (`https://github.com/f-miyu/Plugin.CloudFirestore`); si è scelto di usare questo plugin cross-platform

sviluppato da terzi perchè non esiste alcun plugin cross-platform ufficiale che consenta la gestione del database offerto da Firebase.

Nel Listato 6.7 viene riportata una porzione di codice, presente nel file `HomePage.xaml.cs`, che consente il download dei dati delle ultime dieci ricette approvate.

```
...

private async Task LoadLastTen()
{
    var group = await CrossCloudFirestore.Current.
       Instance.
       GetCollection("Ricette").
       WhereEqualsTo("isApproved", true).
       OrderBy("Timestamp", true).
       LimitTo(10).
       GetDocumentsAsync();
    List<Ricetta> ricette = group.ToObjects<Ricetta>().ToList();
    Ricette = new ObservableCollection<Ricetta>(ricette);
    BindableLayout.SetItemsSource(LastTenRecipes, Ricette);
}

...
```

Listato 6.7. Codice per il caricamento delle ultime dieci ricette, presente nel file `HomePage.xaml.cs`

Nel listato sopra riportato l'applicazione, tramite la prima istruzione, salva nella variabile `group` le ultime dieci ricette pubblicate e converte il suo contenuto in una lista, per poi salvarlo nella lista di ricette `ricette`. Successivamente, essa provvede, nelle modalità specificate, alla visualizzazione dei dati ottenuti.

Nel Listato 6.8 viene riportata una porzione di codice, presente nel metodo `upload` nel file `InsertPage.xaml.cs`, che consente l'inserimento nel database di una ricetta da approvare.

```
...

await CrossCloudFirestore.Current.Instance.GetCollection("Ricette").AddDocumentAsync(ricetta);

...
```

Listato 6.8. Codice per l'upload di una ricetta da approvare, presente nel file `InsertPage.xaml.cs`

Nel Listato 6.9 viene riportata una porzione di codice, presente nel file `ShowPage.xaml.cs`, che consente l'aggiunta o la rimozione della ricetta visualizzata dai preferiti.

```
...

 private async void multiFabAction(object sender, EventArgs e)
{

   ...

      if (!CrossFirebaseAuth.Current.Instance.CurrentUser.IsAnonymous)
      {
          if (isFav)
          {
              isFav = false;
              utente.Preferiti.Remove(r.Id);
              await CrossCloudFirestore.Current.Instance.GetCollection("Utenti").GetDocument(utente.Id).
              UpdateDataAsync("Preferiti", utente.Preferiti);
              multiFab.ImageSource = "favourite";
              DependencyService.Get<IAndroidPopUp>().ShowSnackbar("Rimosso dai preferiti");

          }
          else
          {
              isFav = true;
              utente.Preferiti.Add(r.Id);
```

```
                await CrossCloudFirestore.Current.Instance.GetCollection("Utenti").GetDocument(utente.Id).
                UpdateDataAsync("Preferiti", utente.Preferiti);
                multiFab.ImageSource = "favourite_full";
                DependencyService.Get<IAndroidPopUp>().ShowSnackbar("Aggiunto ai preferiti");

            }
        }

    ...
}


...
```

Listato 6.9. Codice presente nel file `ShowPage.xaml.cs` per l'inserimento o la rimozione di una ricetta dai preferiti

Nella porzione di codice riportata nel Listato 6.9, l'applicazione controlla che l'utente non sia anonimo; a seconda del valore presente nella variabile booleana `isFav`, impostata precedentemente dall'applicazione su `false`, se la ricetta non era una ricetta preferita, aggiorna sia il valore della variabile stessa che il il vettore *Preferiti* in Firebase; successivamente cambia l'icona del pulsante dei preferiti; infine, avvisa l'utente, tramite una *snackbar*, dell'aggiunta/rimozione della ricetta dai preferiti.

Flutter

Nell'implementazione in Flutter, per poter effettuare interazioni con il database online, si è usato il pacchetto `cloud_firestore` (`https://pub.dev/packages/cloud_firestore`), supportato ufficialmente dagli sviluppatori di Firebase.

Nel Listato 6.10 viene riportata una porzione di codice, presente nel file `home_screen.dart`, che consente il download dei dati delle ultime dieci ricette approvate.

```
...

void loadLastTenRecepies() {
_lastTenRicette.clear();
CollectionReference recipesCollection =
    FirebaseFirestore.instance.collection("Ricette");
recipesCollection
    .orderBy("Timestamp", descending: true)
    .where("isApproved", isEqualTo: true)
    .limit(10)
    .get()
    .then((QuerySnapshot querySnapshot) async {
  for (int i = 0; i < querySnapshot.docs.length; i++) {
    Ricetta ricetta = new Ricetta(
        querySnapshot.docs[i].id,
        querySnapshot.docs[i].get("Autore"),
        querySnapshot.docs[i].get("Thumbnail"),
        querySnapshot.docs[i].get("Titolo"),
        querySnapshot.docs[i].get("NumeroPersone"),
        querySnapshot.docs[i].get("TempoPreparazione"),
        List<Map<String, dynamic>>.from(
            querySnapshot.docs[i].get("Ingredienti")),
        List<String>.from(querySnapshot.docs[i].get("Passaggi")),
        querySnapshot.docs[i].get("isApproved"),
        querySnapshot.docs[i].get("Timestamp"),
        querySnapshot.docs[i].get("Categoria"));
    _lastTenRicette.add(ricetta);
  }
  setState(() {
    _lastTenRicetteLoaded = true;
  });
});
}

...
```

Listato 6.10. Codice per il caricamento delle ultime dieci ricette presente nel file `home_page.dart`

Nel listato sopra riportato, l'applicazione, tramite la terza istruzione, legge le ultime dieci ricette pubblicate una alla volta e le salva, mano a mano, nel vettore di ricette `_lastTenRicette`.

Nel Listato 6.11 viene riportata una porzione di codice, presente nel metodo `uploadToFirebase` del file `insert_ricetta_screen.dart`, che consente l'inserimento nel database di una ricetta da approvare.

```
...

FirebaseFirestore.instance
        .collection("Ricette")
        .add(ricetta)
        .then((value) async {
      Navigator.of(context).pop();
      await buildSuccessDialog();
      Navigator.of(context).pop();
    }).onError((error, stackTrace) async {
      await buildUploadErrorDialog();
    });

...
```

Listato 6.11. Codice, presente nel metodo `uploadToFirebase` del file `insert_ricetta_screen.dart`, per l'upload di una ricetta da approvare

Nel Listato 6.12 viene riportata una porzione di codice, presente nel file `ricetta_show_screen.dart`, che consente l'aggiunta o la rimozione della ricetta visualizzata dai preferiti.

```
...

  void _favoriteManagement() {
    DocumentReference actualUserFav =
        FirebaseFirestore.instance.collection("Utenti").doc(_actualUser.id);
    if (!isFav) {
      _actualUser.preferiti.add(_ricetta.id);
      actualUserFav.update({'Preferiti': _actualUser.preferiti}).then((value) {
        setState(() {
          favIcon = Icon(Icons.favorite);
          isFav = !isFav;
        });
        ScaffoldMessenger.of(context)
            .showSnackBar(SnackBar(content: Text("Aggiunto ai preferiti")));
      }).onError((error, stackTrace) {
        _actualUser.preferiti.remove(_ricetta.id);
      });
    } else {
      _actualUser.preferiti.remove(_ricetta.id);
      actualUserFav.update({'Preferiti': _actualUser.preferiti}).then((value) {
        setState(() {
          favIcon = Icon(Icons.favorite_border);
          isFav = !isFav;
        });
        ScaffoldMessenger.of(context)
            .showSnackBar(SnackBar(content: Text("Rimosso dai preferiti")));
      }).onError((error, stackTrace) {
        _actualUser.preferiti.add(_ricetta.id);
      });
    }
  }

...
```

Listato 6.12. Codice, presente nel file `ricetta_show_screen.dart`, per l'aggiunta/rimozione di una ricetta dai preferiti

L'applicazione, dopo aver controllato che l'utente non sia anonimo, carica nell'interfaccia grafica un pulsante che, se premuto, chiamerà il metodo `_favoriteManagement`, presente nel Listato 6.12. Il metodo citato pocanzi utilizza la variabile booleana `isFav`, impostata su `true` se la ricetta è una di quelle preferite dell'utente. Se l'utente, ad esempio, clicca il pulsante per aggiungere la ricetta ai preferiti, l'ap-

plicazione aggiorna sia il valore del vettore `preferiti`, presente in memoria, che il vettore *Preferiti* in Firebase. Successivamente essa cambia sia l'icona del pulsante con un cuore pieno, che il valore della variabile `isFav` a `true`, per notificare infine all'utente, tramite una *snackbar*, che la ricetta è stata aggiunta ai preferiti.

6.2.3 Storage

Xamarin

Per poter effettuare interazioni con il servizio Firebase Storage nell'implentazione in Xamarin, si è usato il plugin `Plugin.FirebaseStorage` (`https://github.com/f-miyu/Plugin.FirebaseStorage`); si è scelto di utilizzare questo plugin cross-platform sviluppato da terzi perchè non esiste alcun plugin cross-platform ufficiale che consenta la gestione delle interazioni con Firebase Storage.

Nel Listato 6.13 viene riportata una porzione di codice, presente nel file `InsertPage.xaml.cs`, che consente la gestione dell'upload di un'immagine su Firebase Storage.

```
...
private async void upload()
{
    ...
    string filename = Guid.NewGuid().ToString() + ".jpg";
                var reference = CrossFirebaseStorage.Current.Instance.RootReference.GetChild(filename);
                var uploadProgress = new Progress<IUploadState>();
                uploadProgress.ProgressChanged += (sender, e) =>
                {
                    var progress = e.TotalByteCount > 0 ? 100.0 * e.BytesTransferred / e.TotalByteCount : 0;
                };
                await reference.PutStreamAsync(imgFile.GetStream(), progress: uploadProgress);
                ricetta.Thumbnail = (await reference.GetDownloadUrlAsync()).ToString();
        ...
}
...
```

Listato 6.13. Codice per la gestione dell'upload dell'immagine, presente nel file `InsertPage.xaml.cs`

Come si può verificare nel listato sopra riportato, l'applicazione genera il nome del file con un `UUID` e, nelle istruzioni successive, provvede ad effettuare l'upload dell'immagine in Firebase Storage; infine, memorizza in `ricetta.Thumbnail` il link ricollegato all'immagine appena caricata online attraverso l'ultima istruzione.

Flutter

Nell'implementazione in Flutter, per poter effettuare interazioni con il servizio Firebase Storage, si è usato il pacchetto `firebase_storage` (`https://pub.dev/packages/firebase_storage`), supportato ufficialmente dagli sviluppatori di Firebase.

Nel Listato 6.14 viene riportata una porzione di codice, presente nel file `insert_ricetta_screen.dart`, che consente la gestione dell'upload di un'immagine in Firebase Storage.

```
...
void uploadToFirebase(Map ricetta) async {
    var uuid = Uuid().v4();
    Reference ref = FirebaseStorage.instance.ref();
    Reference img = ref.child(uuid.toString() + ".jpg");
```

```
        buildUploadDialog("Upload della ricetta...");
        await img.putFile(f);
        await img.getDownloadURL().then((url) {
          ricetta["Thumbnail"] = url;
          ...
        }
    }
    ...
```

Listato 6.14. Codice per la gestione dell'upload dell'immagine presente nel file `insert_ricetta_screen.dart`

Come si può verificare nel listato sopra riportato, l'applicazione genera il nome del file con un `UUID` e, nelle istruzioni successive, provvede ad effettuare l'upload dell'immagine in Firebase Storage; infine, memorizza in `ricetta["Thumbnail"]` il link ricollegato all'immagine appena caricata online attraverso l'ultima istruzione.

6.3 Peso degli APK

L'estensione `APK` indica un file "Android Package", ovvero una variante del formato `.jar`, utilizzata per la distribuzione e l'installazione di applicazioni per Android.

Durante la fase di debug, la variante di `APK` utilizzata è quella di `debug`, dove il pacchetto sarà firmato con una chiave di default e con il flag di debug abilitato.

Per consentire l'installazione su tutti i dispositivi sarà necessario generare l'`APK` nella variante `release`; il flag di debug è disattivato, in modo che il pacchetto non possa essere sottoposto a debug e, inoltre, lo sviluppatore deve indicare una propria chiave per la firma del pacchetto.

Nelle Figure 6.3 e 6.4 sono riportate, rispettivamente per Xamarin e Flutter, le informazioni riguardanti i file `APK` nella variante di `release`.

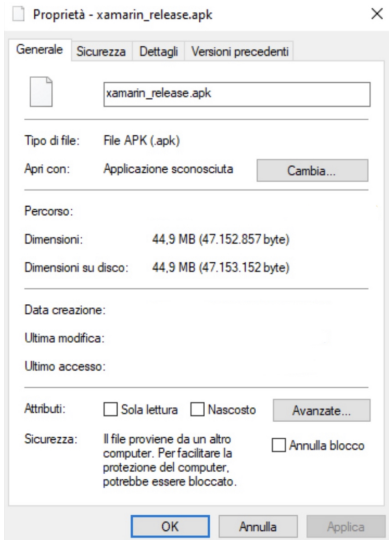

Figura 6.3. Proprietà del file `APK` creato tramite Xamarin

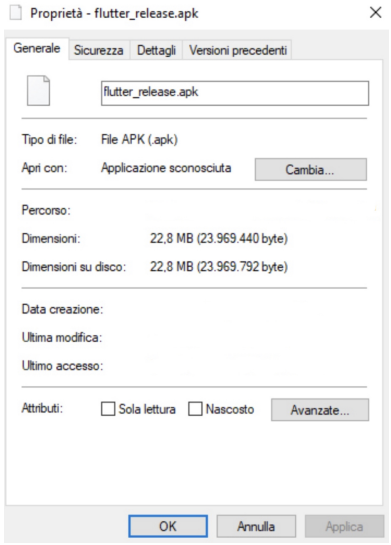

Figura 6.4. Proprietà del file `APK` creato tramite Flutter

Come si può evincere dalle figure precedenti, il pacchetto `APK` creato tramite Xamarin (con un peso di 44.9 MB) ha dimensione quasi doppia rispetto a quello generato tramite Flutter (con un peso di 22.8 MB).

7

Conclusioni

In questo studio è stata realizzata la stessa social app sia attraverso il framework Flutter che mediante il framework Xamarin, ovvero tramite due linguaggi cross-platform differenti, e sono state, quindi, messe a confronto le due implementazioni.

Per garantire una maggiore chiarezza della tematica di studio, il primo passo affrontato è stato quello di introdurre il framework Flutter, approfondendo inizialmente le modalità di installazione e, successivamente, spiegando la procedura per la creazione di un nuovo progetto; infine, sono stati spiegati i concetti chiave del funzionamento dell'applicazione iniziale generata.

Successivamente è stato approfondito il secondo framework, ossia Xamarin, approfondendo inizialmente le modalità di installazione e, successivamente, spiegando la procedura per la creazione di un nuovo progetto; infine, sono stati illustrati i concetti chiave del funzionamento dell'applicazione iniziale generata.

Una volta terminata questa fase introduttiva, è stata dapprima effettuata l'analisi dei requisiti, funzionali e non, per poi entrare nel dettaglio della progettazione dell'app mostrando la struttura dell'applicazione e, successivamente, spiegando, tramite l'uso dei flowchart, il funzionamento della stessa in alcune situazioni. Sono stati, poi, creati i vari mockup dell'interfaccia utente e, infine, è stata effettuata la progettazione della componente dati dove sono stati utilizzati alcuni servizi di Firebase per consentire sia l'autenticazione dell'utente, sia la gestione dei dati dello stesso e delle ricette, sia, infine, la gestione delle immagini.

Nella fase successiva è stata implementata l'applicazione in Xamarin, cercando di ottenere una User Interface piacevole e fluida.

Terminata l'implementazione in Xamarin, è stata effettuata l'implementazione della stessa applicazione in Flutter.

Nell'ultima fase sono state messe a confronto le due implementazioni, esponendo, nella prima parte, alcune differenze nell'interfaccia grafica, per poi, nella seconda parte, confrontare le varie modalità di accesso ai servizi offerti da Firebase; nella terza ed ultima parte sono state messe a confronto le dimensioni dei file APK.

Riteniamo che le conclusioni a cui siamo giunti con il presente lavoro debbano costituire un punto di partenza per sviluppi futuri, in linea con gli obiettivi che ci eravamo posti già nell'introduzione. Sebbene l'applicazione creata possegga già diverse funzionalità, potrebbero esserne implementate altre, come la ricerca per ingredienti, la condivisione rapida delle ricette tramite link, la visualizzazione di

pubblicità tramite Google AdMob e, infine, la revisione automatica delle ricette da approvare. Potrebbero, inoltre, essere aggiunte ulteriori modalità di autenticazione nella fase di accesso all'app, come quella tramite Facebook, Instagram e Twitter.

Riferimenti bibliografici

1. Installazione di Xamarin. `https://docs.microsoft.com/it-it/xamarin/get-started/installation/?pivots=windows`, 2021.
2. Windows install - Flutter. `https://flutter.dev/docs/get-started/install/windows`, 2021.
3. S. Alessandria. *Crea la Tua App con Flutter: Guida Pratica per Creare App per Android, iOS e il Web con Dart e Flutter*. Independently published, 2020.
4. S. Alessandria. *Flutter Projects: A practical, project-based guide to building real-world cross-platform mobile applications and games*. Packt Publishing, 2020.
5. J. Bach, C. Alves, and R. Stewart. *Xamarin: Xamarin for beginners, Building Your First Mobile App with C# .NET and Xamarin*. Independently published, 2021.
6. A. Biessek. *Flutter for Beginners: An introductory guide to building cross-platform mobile applications with Flutter and Dart 2*. Packt Publishing, 2019.
7. D. Hermes. *Xamarin Mobile Application Development: Cross-Platform C# and Xamarin.Forms Fundamentals*. Apress, 2015.
8. D. Hermes. *Building Xamarin.Forms Mobile Apps Using XAML: Mobile Cross-Platform XAML and Xamarin.Forms Fundamentals*. Apress, 2019.
9. D. Hindrikes, J. Karlsson, and D. Ortinau. *Xamarin.Forms Projects: Build multiplatform mobile apps and a game from scratch using C# and Visual Studio 2019*. Packt Publishing, 2020.
10. M. Katz, K.D. Moore, and V. Ngo. *Flutter Apprentice (First Edition): Learn to Build Cross-Platform Apps*. Razeware LLC, 2021.
11. A. Miola. *Flutter Complete Reference: Create beautiful, fast and native apps for any device*. Independently published, 2020.
12. M.L. Napoli. *Beginning Flutter: A Hands on Guide to App Development*. Wrox Pr Inc, 2019.
13. R. Payne. *Beginning App Development With Flutter: Create Cross-Platform Mobile Apps*. Apress, 2019.
14. C. Petzold. *Creating Mobile Apps with Xamarin.Forms Preview Edition 2*. Microsoft Press, 2015.
15. E. Snider and D. Ortinau. *Mastering Xamarin.Forms: App architecture techniques for building multi-platform, native mobile apps with Xamarin.Forms 4*. Packt Publishing, 2019.
16. G. Versluis and S. Thewissen. *Xamarin.Forms Solutions*. Apress, 2018.
17. Wikipedia. Flutter (software). `https://en.wikipedia.org/wiki/Flutter_(software)`, 2021.
18. Wikipedia. Universally unique identifier. `https://en.wikipedia.org/wiki/Universally_unique_identifier`, 2021.

19. Wikipedia. Xamarin. `https://en.wikipedia.org/wiki/Xamarin`, 2021.
20. E. Windmill and R. Rishpater. *Flutter in Action*. Manning Pubns Co, 2020.
21. C. Zaccagnino. *Flutter. Guida allo sviluppo di app performanti e cross-platform*. Hoepli, 2020.
22. F. Zammetti. *Practical Flutter: Improve your Mobile Development with Google's Latest Open-Source SDK*. Apress, 2019.