Progetto Programmazione Mobile:

BollettApp

Daniele Pallini 1083424
Andrea Giuliani 1085273
Diego Pranzetti 1081491

Indice

Introduzione

Il progetto

Nei mesi passati, durante la quarantena da Covid, abbiamo pensato di creare un qualcosa che potesse essere un minimo utile alla comunità.
Il progetto, realizzato nell'ambito del corso di Programmazione Mobile, consiste nello sviluppo di un'app di gestione bollette, in cui un utente registrato ha la possibilità di inserire nuove bollette, controllare l'andamento di costi e consumi e aggiungere le bollette in scadenza al calendario, in modo da fissare un evento come "reminder", utile ad esempio nel momento in cui si hanno tante cose a cui pensare e le scadenze passano in secondo piano.

Requisiti

Requisiti funzionali

- Il visitatore dell'app deve avere la possibilità di registrarsi.
- Il visitatore dell'app deve avere la possibilità di fare login con e-mail e password.
- L'utente può aggiungere una nuova bolletta allo storico.
- L'utente può eliminare una bolletta aggiunta in precedenza.
- L'utente può visualizzare tutte le bollette aggiunte in precedenza.
- L'utente può modificare la password dell'account.
- L'utente può effettuare logout.
- L'utente può aggiungere una bolletta in scadenza al calendario.
- L'utente può visualizzare grafici basati su costo e consumo delle bollette attualmente presenti nello storico.

Requisiti non funzionali

- L'app dovrà avere un design moderno ed accattivante.
- L'app dovrà essere "user friendly" ed intuitiva.
- L'app dovrà avere delle buone performance ed essere reattiva.
- L'app dovrà essere il più affidabile possibile, evitando crash improvvisi ed errori di varia natura.
- L'app dovrà avere un sistema di autenticazione sicuro e difficilmente attaccabile.
- L'app dovrà avere un sistema di gestione dati in remoto, non legato alla memoria del telefono ma a un server esterno(Firebase).

Progettazione

Struttura dell’app

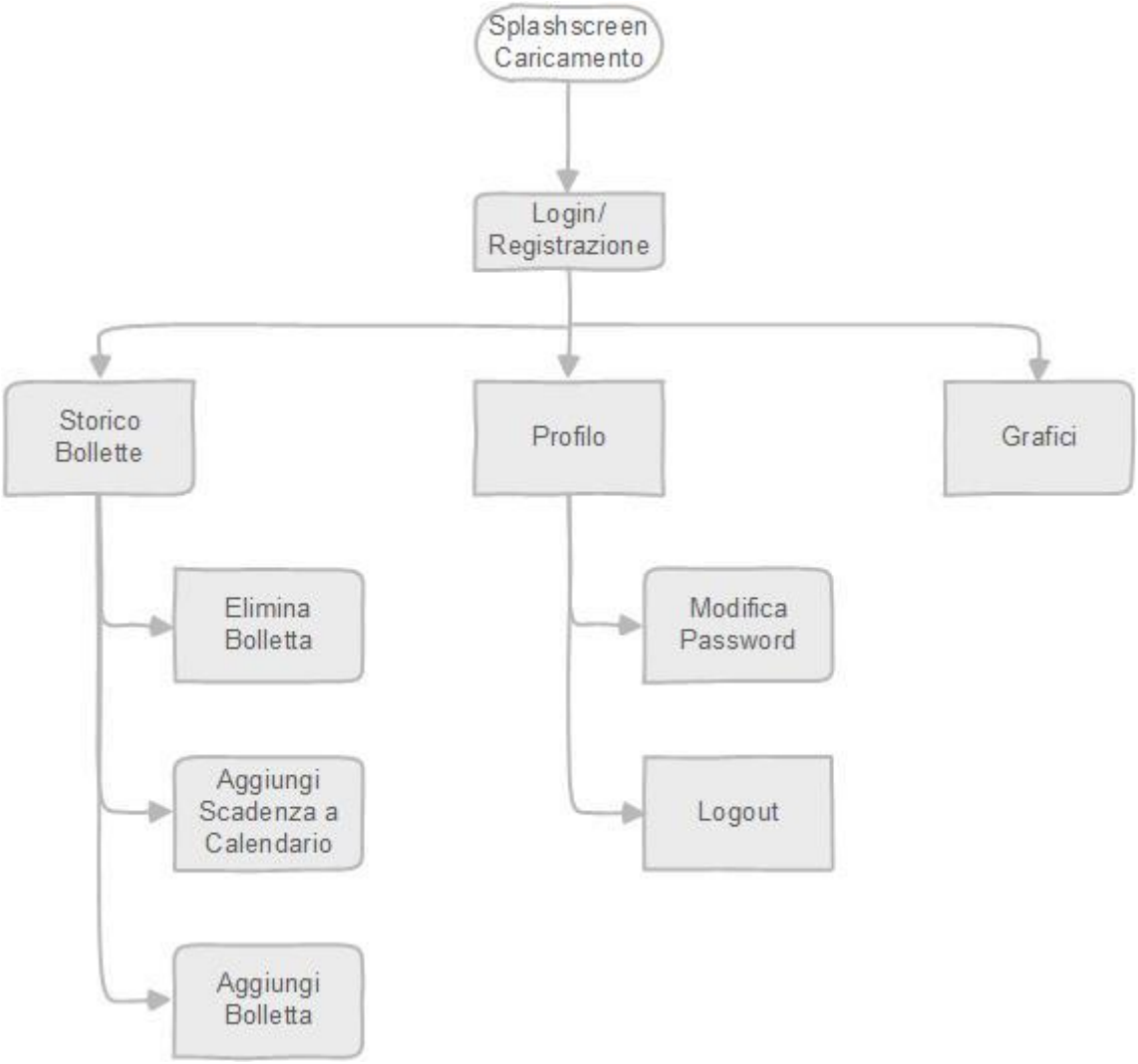


Figura 1: Struttura dell'app.

Flowchart

Di seguito alcuni grafici relativi alle funzionalità principali dell'applicazione, in particolare riguardo Login, inserimento e gestione bollette.

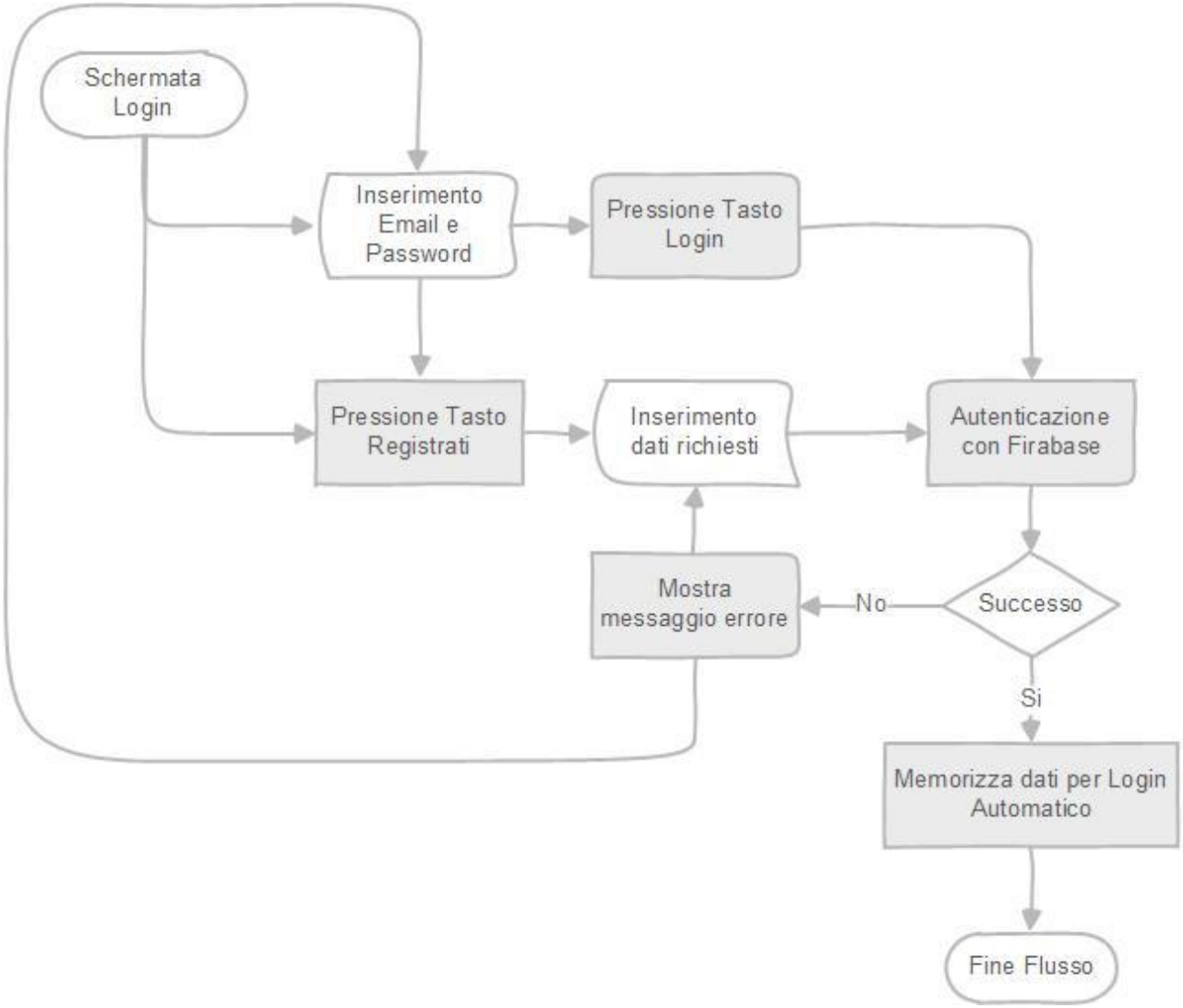


Figura 2: Login.

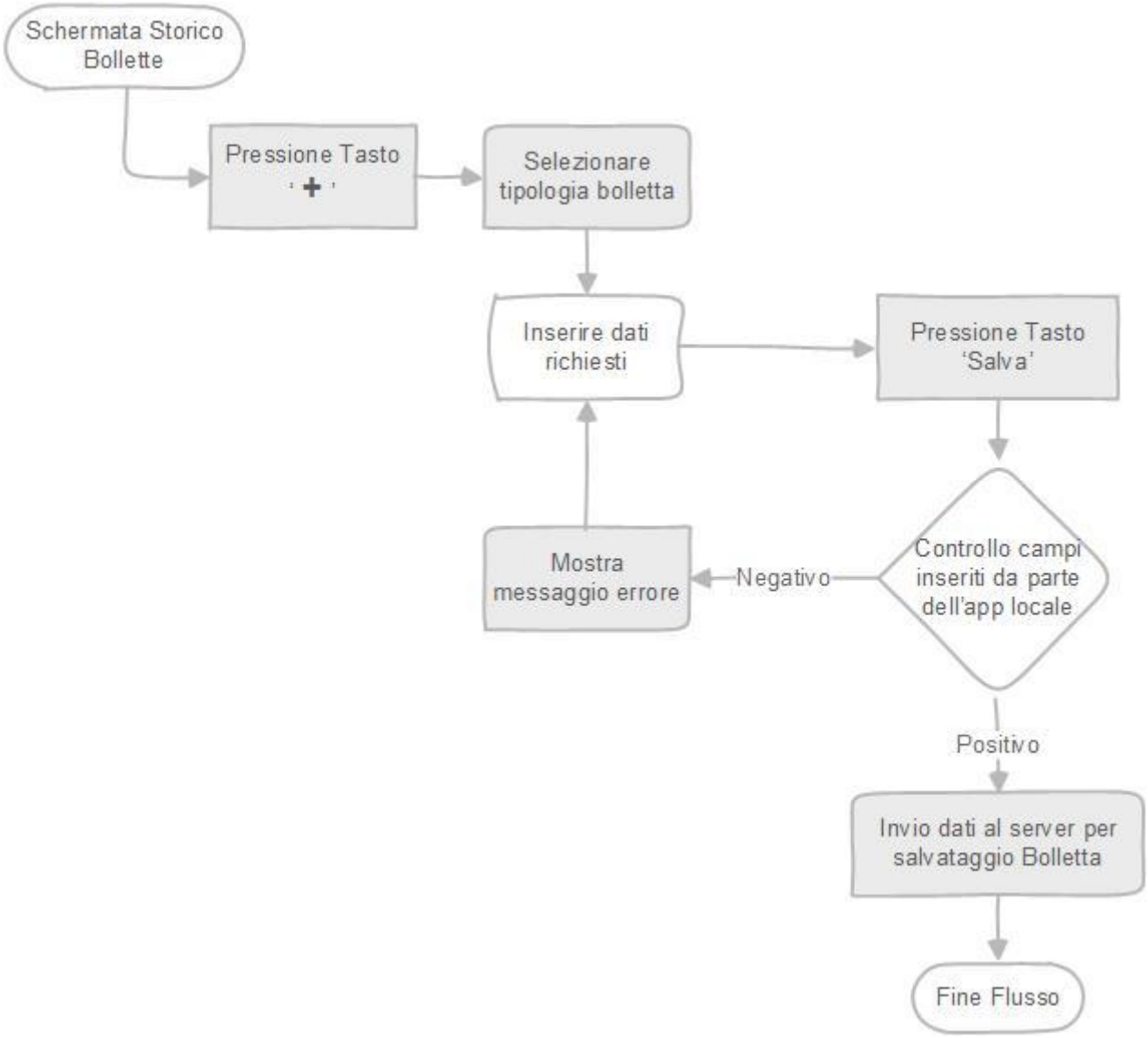


Figura 3: Aggiunta Bolletta.

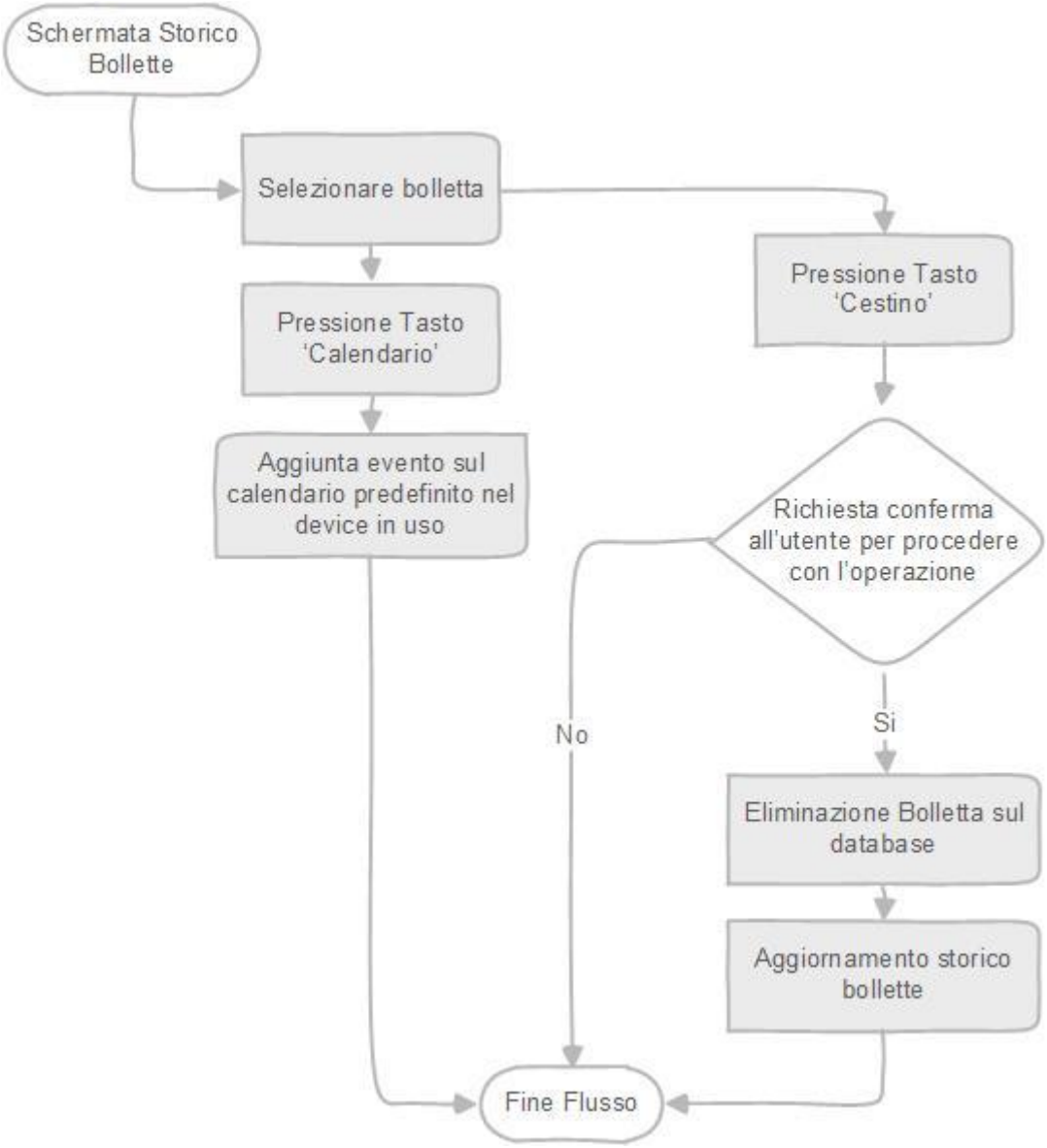
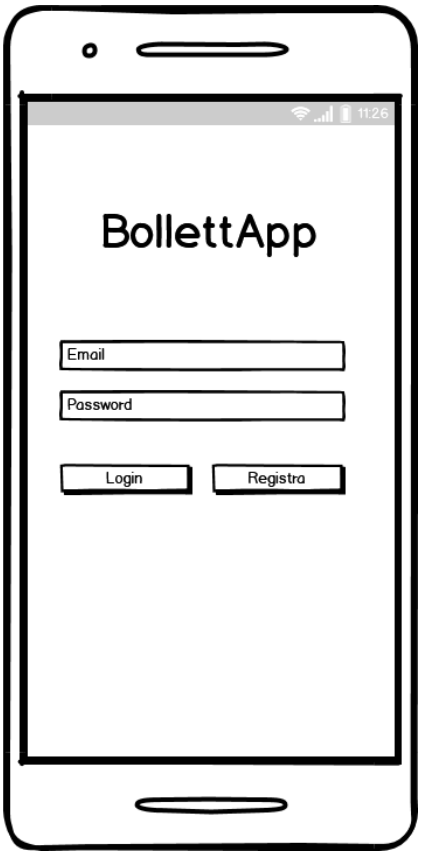


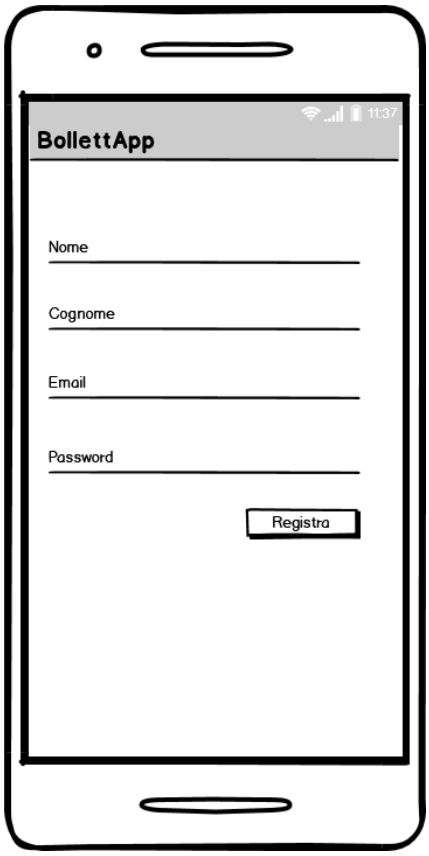
Figura 4: Gestione Bolletta.

Mockups

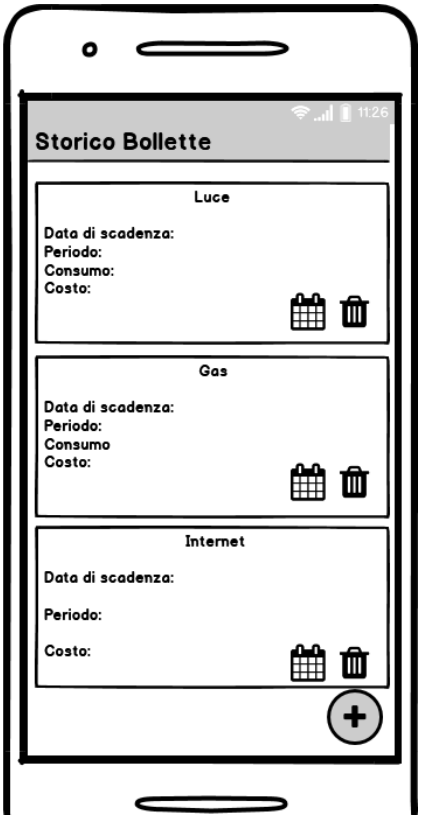
Per la realizzazione dei mockups abbiamo utilizzato la versione di prova dell'applicazione Balsamiq Mockups.
Lo stile grafico utilizzato è il Material Design di Google, che implementa una interfaccia grafica "user-friendly" e minimale, con alcuni aspetti stilisticamente molto interessanti.



Schermata Login.

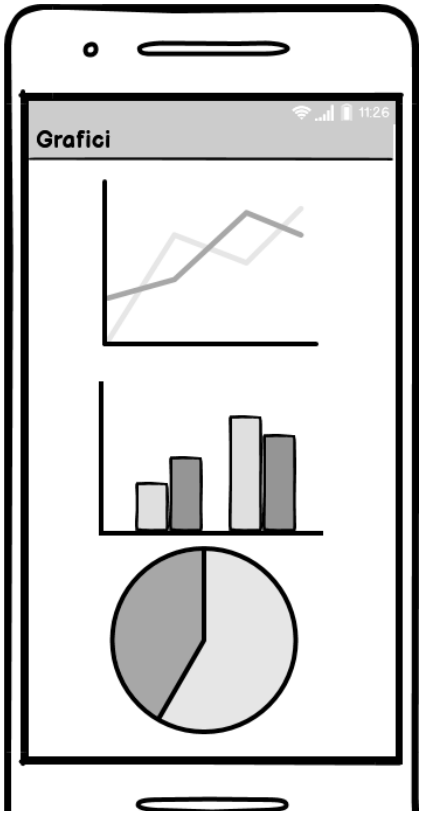


Schermata Registrazione.

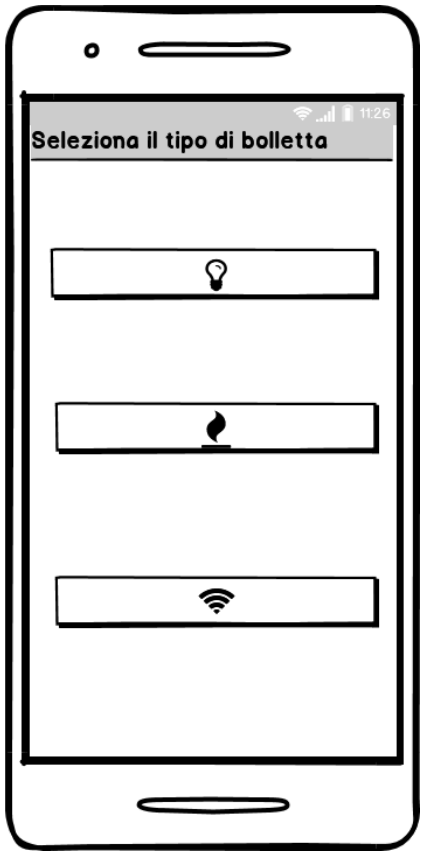


Storico Bollette.

Profilo.



Schermata Grafici.



Selezione Bolletta.

Form Inserimento Bolletta Luce/Gas.

Form Inserimento Bolletta Internet.

Implementazione backend

Firebase

Come server abbiamo deciso di utilizzare Firebase, un servizio di backend realizzato da Google, che offre servizi di database, di autenticazione, di hosting web, di analisi del prodotto e di diagnostica degli utenti, oltre a moltissime altre funzioni.
Il servizio è ben documentato, oltre che largamente supportato da Android, IOS e dalle pagine Web.

Motivi della scelta

I motivi della scelta di Firebase rispetto a un server tradizionale sono molteplici, primo tra tutti il fatto che Firebase è gratuito (almeno nelle funzionalità necessarie per il nostro scopo).
Inoltre, il servizio di database di Firebase, **Firestore** nel nostro caso, ha una struttura semplificata rispetto ai database tradizionali, ad esempio quelli basati su MySQL. Infatti, Firestore non è un database relazionale, ma riunisce documenti in collezioni, accessibili senza nessun tipo di chiave primaria o esterna ma semplicemente tramite nome del documento.
Questi documenti sono in un formato molto simile al JSON, quindi costituiti da coppie chiave-valore, a cui si accede semplicemente con il metodo getString della chiave.
Fatto non indifferente è la sicurezza, un conto è lavorare con server sviluppati e manutenuti da Google, un altro è usarne uno “personale” e con bassissimi livelli di sicurezza informatica.
Infine, Firebase implementa meccanismi di gestione autenticazione e autorizzazione, molto utili nel nostro caso specifico perché ci ha permesso di lasciargli la gestione degli account e di autorizzare login e registrazione, rendendo tutto il processo decisamente più sicuro e affidabile.

Implementazione database

La struttura del database da noi implementata prevede due collezioni di documenti una dentro l'altra.
Nello strato più esterno c'è la collezione utenti.
Ogni documento all'interno di questa collection è rappresentato da un user id(una stringa di caratteri generati automaticamente da firebase) e identifica il singolo utente registrato, caratterizzato dai campi nome e cognome (Figura 1).

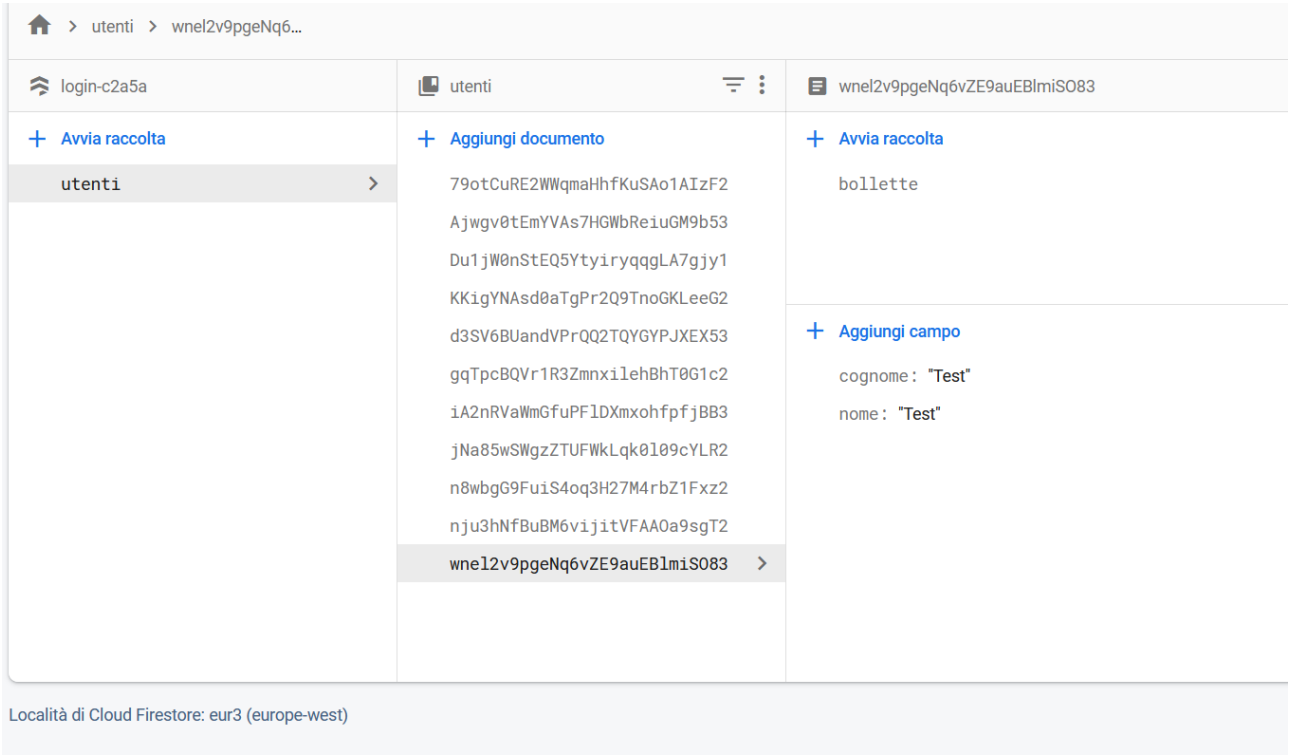


Figura 5: Collezione utenti e documenti utente.

Ogni documento utente ha anche al suo interno un'altra collezione, **bollette**, che contiene tutte le bollette registrate dall'utente stesso, cioè quelle associate al suo profilo (Figura2).
Questo livello di annidamento è stato funzionale a creare una sorta di dipendenza tra bollette e utente che le ha inserite e quindi ci ha reso più immediato il recupero delle bollette in fase di presentazione.

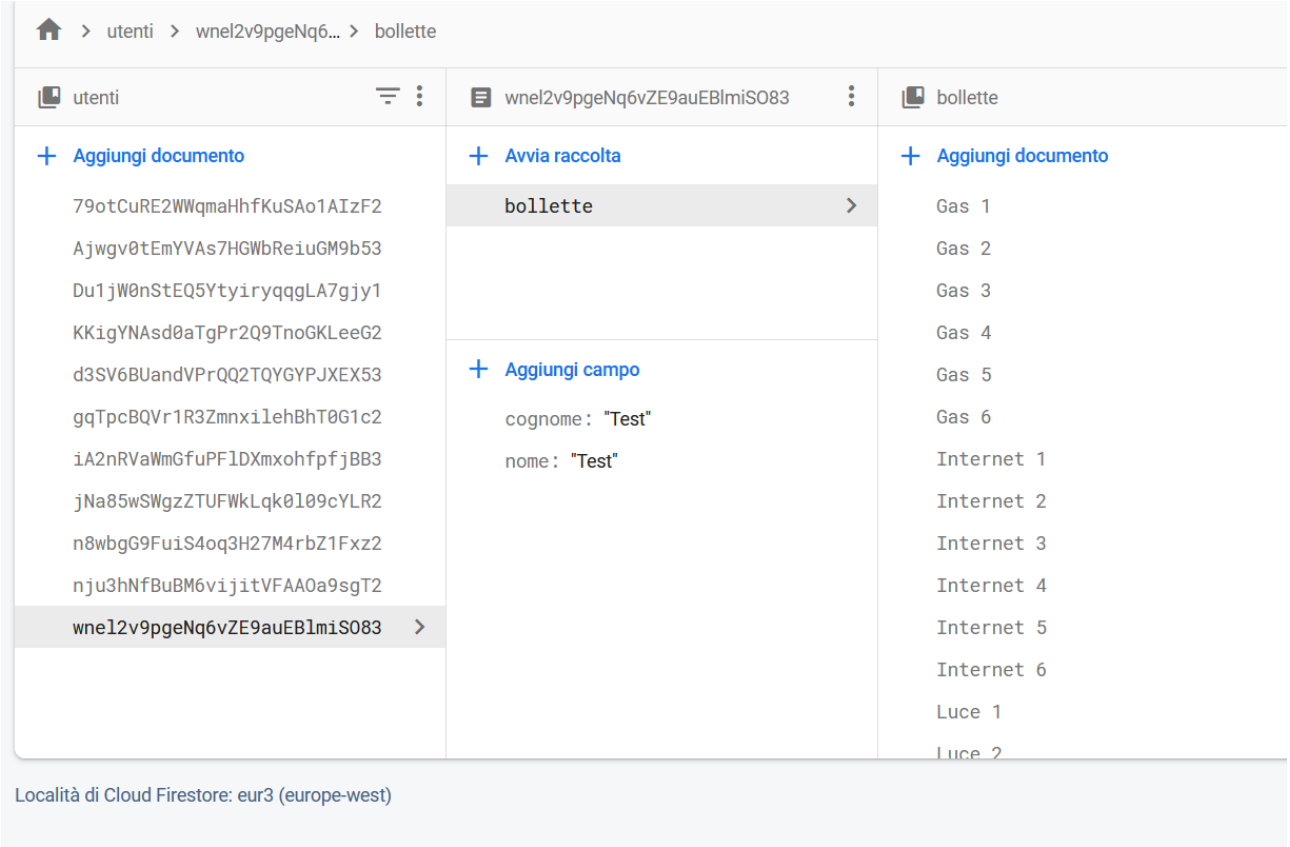


Figura 6: Collezione bollette

La collezione “bollette” presenta al suo interno tutte le bollette registrate dall’utente.
Ciascuna bolletta è caratterizzata da un identificatore univoco costituito dal tipo (“Luce”, “Gas”, “Internet”) e da un codice auto-incrementale per i tre tipi disponibili.
Ad esempio, in Figura 2, Gas 1 rappresenta la prima bolletta di tipo Gas aggiunta dall’utente Test Test.
Entrando più nel dettaglio, in Figura 3 si mostra la bolletta Gas 1.
I campi associati sono:

- ***Tipo:*** *È il tipo della bolletta, costituisce la prima parte del titolo del documento*
- ***Codice:*** *Rappresenta il codice univoco della bolletta, costituisce la seconda parte del titolo del documento.*
- ***DataScadenza***
- ***Da:*** *È l’inizio del periodo di riferimento della bolletta.*

- ***A:*** *È la fine del periodo di riferimento della bolletta.*
- ***Importo***
- ***Consumo***
- ***Misura:*** *Usiamo questo campo per indicare l'unità di misura utilizzata, serve soprattutto per l'app Xamarin.*

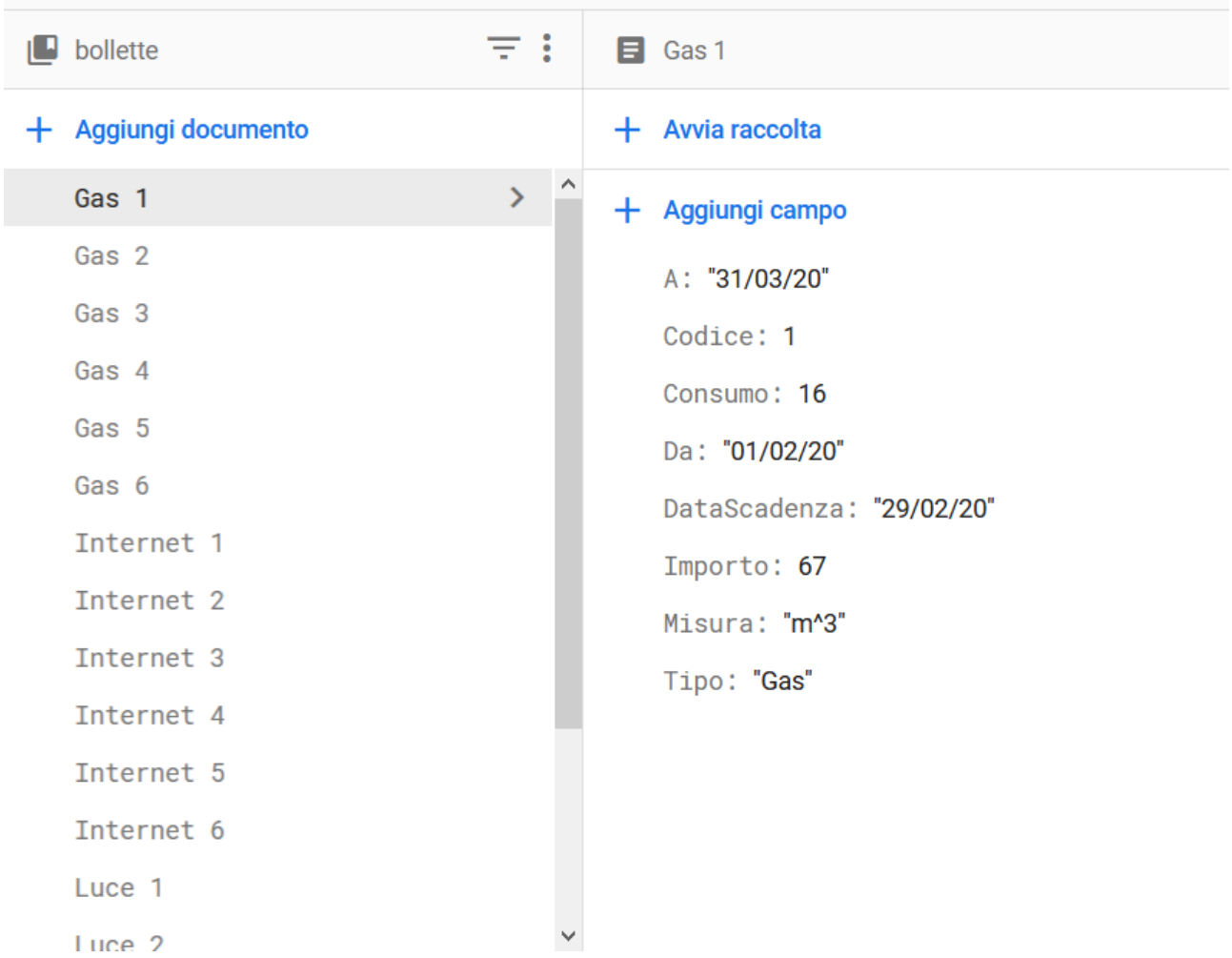


Figura 7: Esempio Bolletta Gas

In modo analogo sono salvate le bollette di tipo Luce e Internet, di seguito un esempio per le due (Figure 4 e 5).
Nota: la bolletta di tipo Luce non ha ovviamente i campi Misura e Consumo.

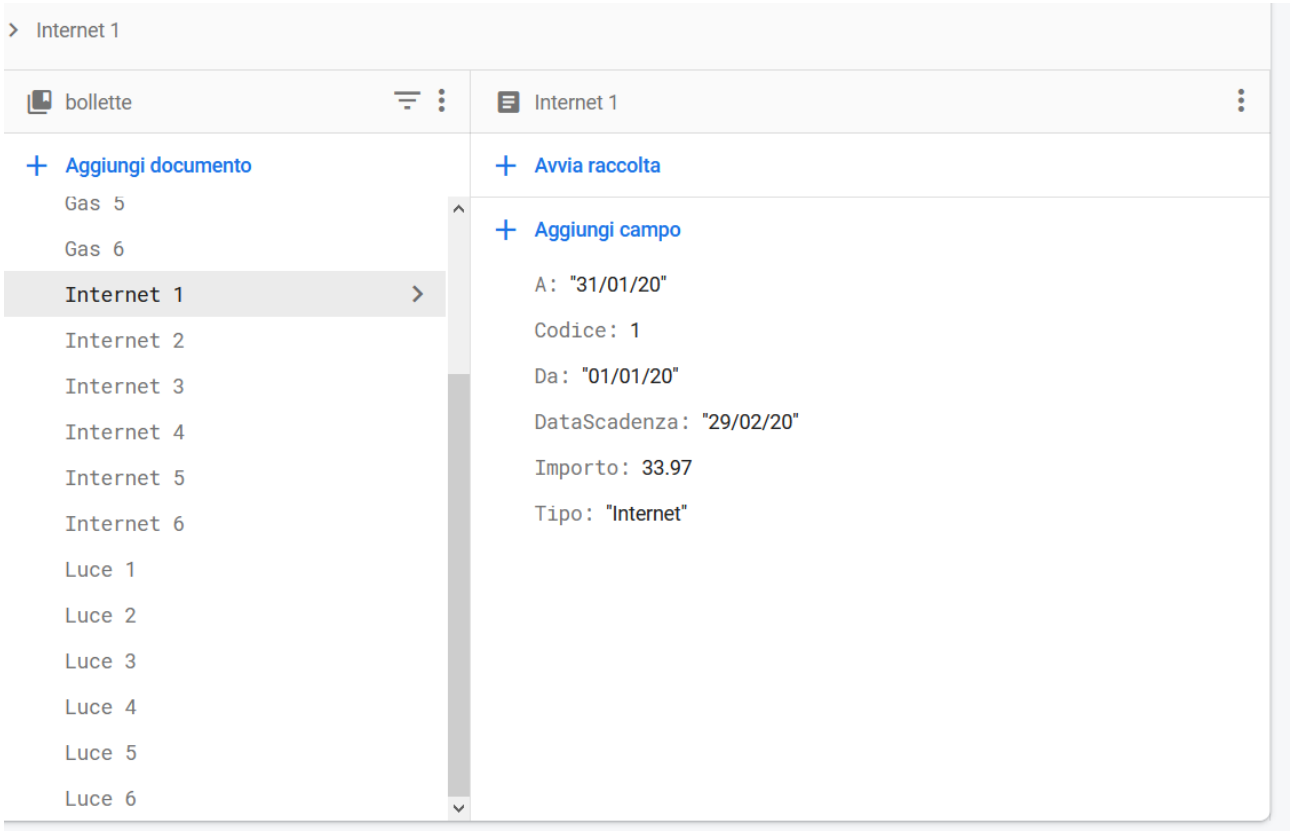


Figura 8: Esempio Bolletta Luce

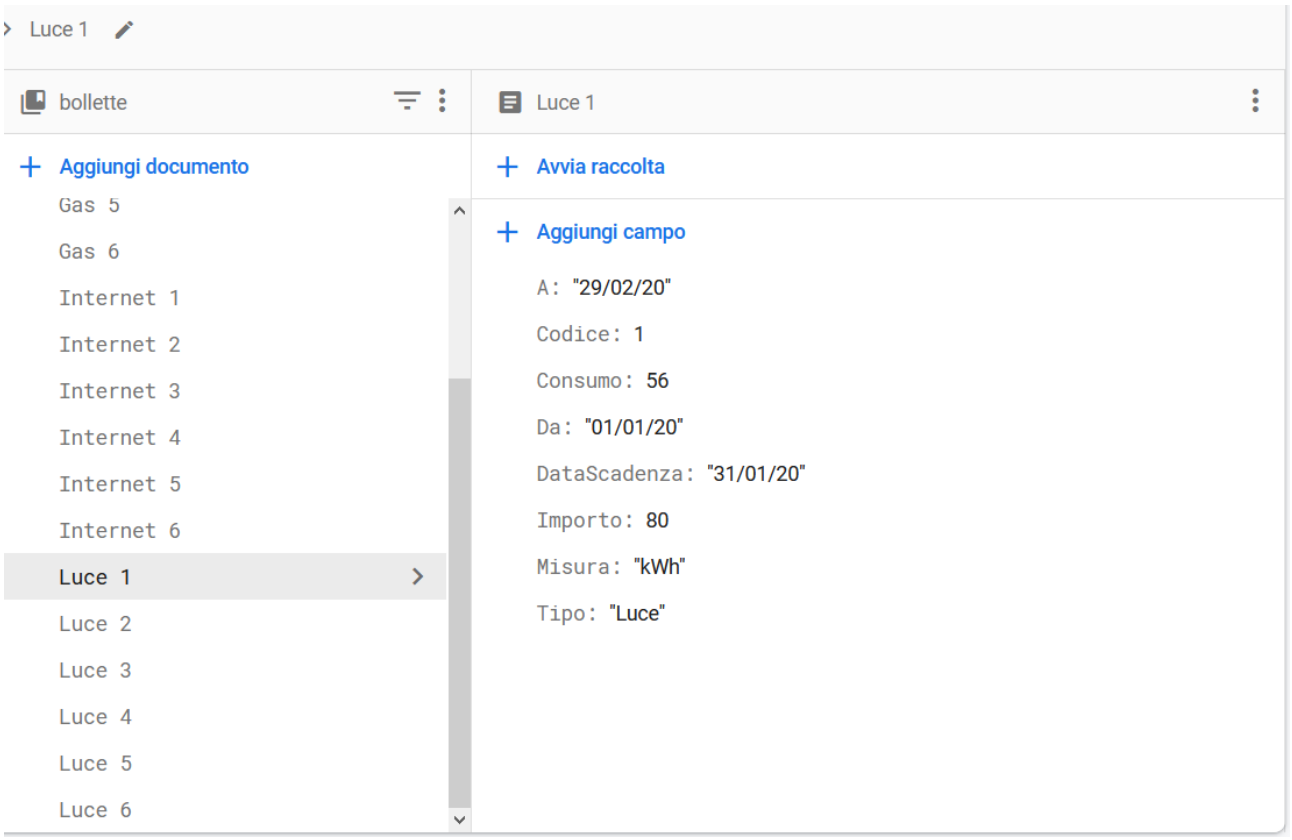


Figura 9: Esempio Bolletta Internet

Storage

Firebase è “all-inclusive”, cioè funge anche da server FTP per lo storage dei file. Per questo ci siamo serviti di Firebase **Storage**, che lavora in maniera simile ad un cloud storage, cioè identificando i file come URL, recuperati poi nella nostra app per il caricamento dell’immagine profilo dell’account utente.

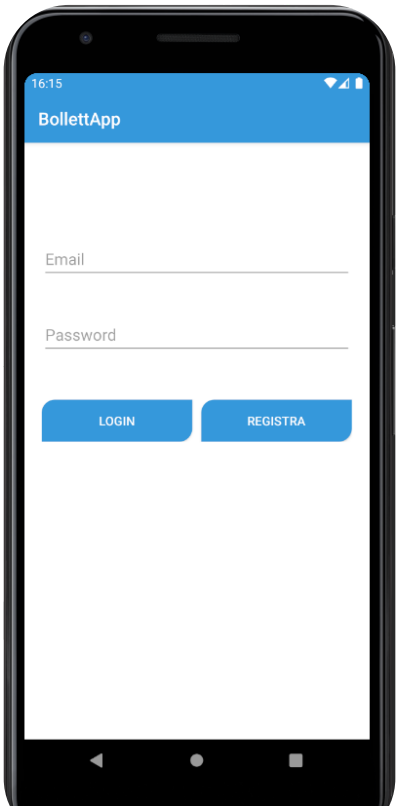
Implementazione Android

Librerie e Plugin utilizzati

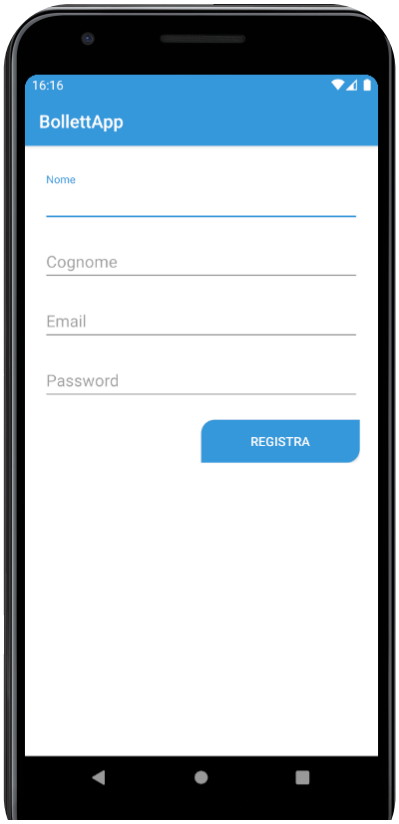
Di seguito sono riportate le librerie utilizzate nel progetto di Android Studio.

- Firebase core, Firebase Firestore, Firebase Storage: Librerie per integrare il database di firebase nell'applicazione Android.
- Firebase Auth: Libreria per l'implementazione del processo di autenticazione.
- Material: Libreria che implementa le componenti grafiche del Material Design.
- Glide: Libreria per la gestione delle immagini.
- MPAndroidChart: Libreria per implementazione e visualizzazione grafici.

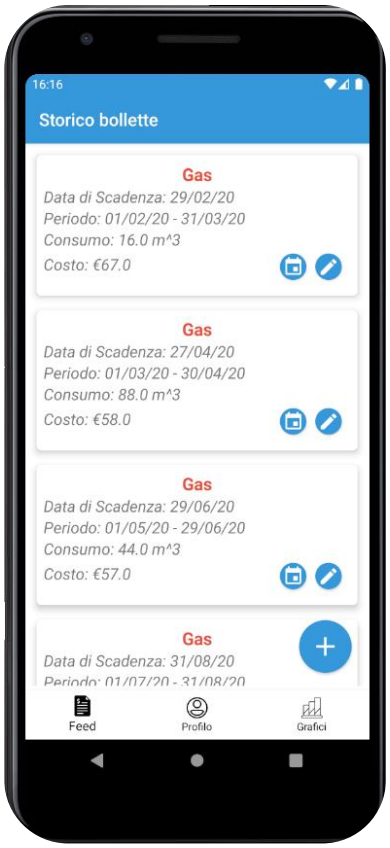
Screenshots:



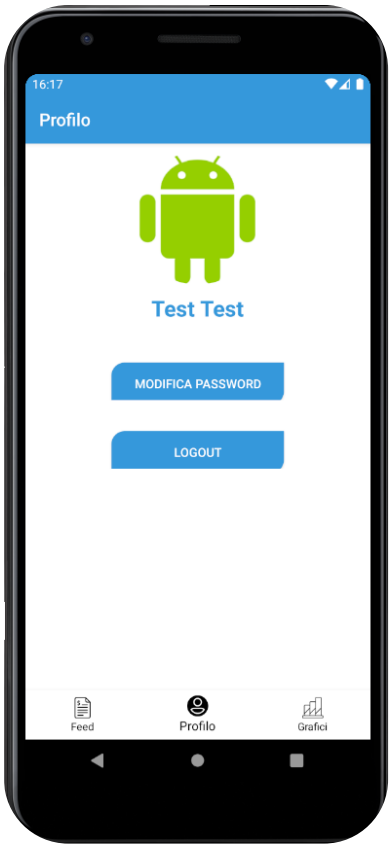
Login



Registrazione



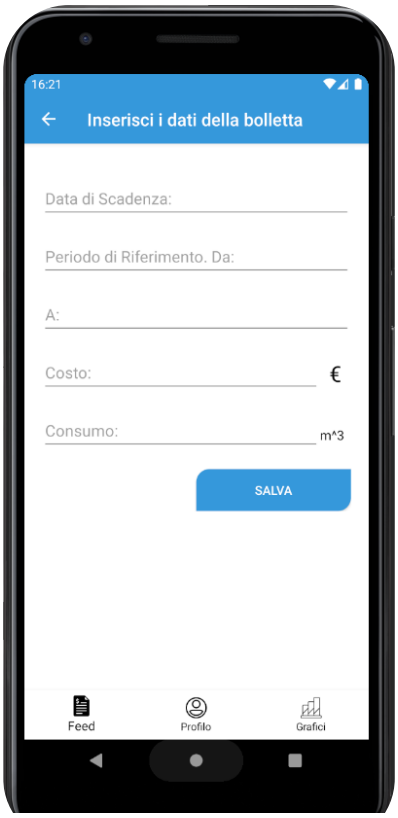
Storico Bollette



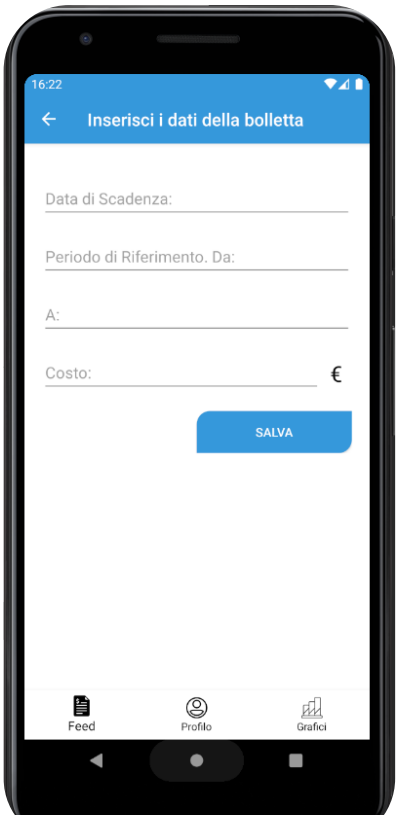
Profilo

Grafici

Selezione Bolletta



Form Inserimento Bolletta Luce/Gas



Form Inserimento Bolletta Internet

Note:

I grafici presenti serviranno all'utente per farsi un'idea dell'andamento dei costi in modo chiaro e veloce:

- Il grafico a linee mostra la divisione per categoria identificando ogni bolletta singolarmente(in base al Codice) così da avere una vista generale sull'andamento dei costi.
- Il grafico a barre somma i costi sostenuti in una mensilità di ogni tipo di bolletta.
- Il grafico a torta unisce tutte le bollette dividendole per tipo così da poter capire l'influenza di ogni categoria sul totale dei costi.

In Xamarin il grafico a linee è sostituito da un grafico a radar, che mostra anch'egli l'andamento dei costi mensili.

La modifica della password per motivi di sicurezza interni a Firebase può essere fatta solo se è stato effettuato il login di recente, altrimenti verrà visualizzato un messaggio di errore. Si invita in questo caso ad effettuare logout ed accedere di nuovo all'account.

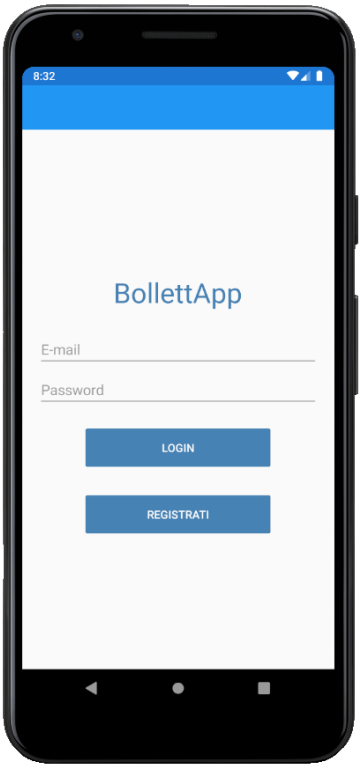
Implementazione Xamarin

Librerie e plugin NuGet utilizzati

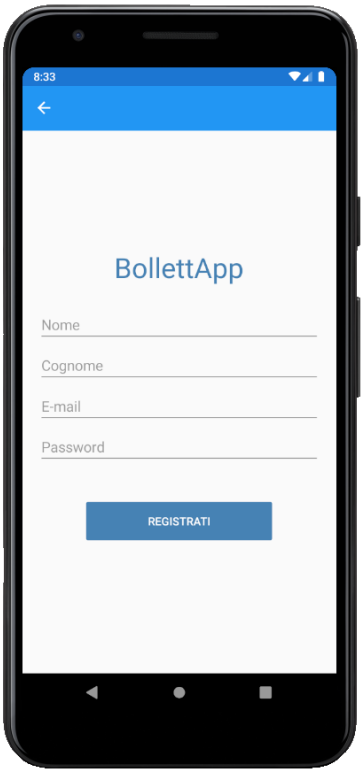
Di seguito sono riportati, con una breve descrizione delle loro funzionalità, i pacchetti utilizzati nel progetto cross platform.

- Plugin.CloudFirestore: plugin cross-platform per effettuare query sul database Firestore.
- Xamarin.Firebase.Core, Xamarin.Firebase.Common: plugin per implementare le funzionalità base di Firebase.
- Xamarin.Firebase.Auth: plugin per implementare la gestione delle autorizzazioni(come il login).
- Xamarin.Essentials: plugin con una serie di API per la grafica interessanti.
- Microcharts, Microcharts.Forms: plugin cross-platform per implementare grafici.
- CClarke.plugin.Calendars: plugin cross-platform per effettuare query e modificare eventi sulle applicazioni calendario del dispositivo.
- Xamarin.GooglePlayServices.Base,Xamarin.GooglePlayServices.Basement: plugin necessari per il funzionamento dei plugin di Firebase.

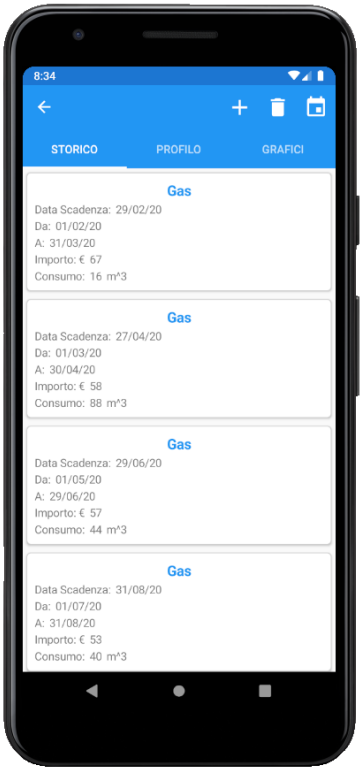
Screenshots Xamarin:



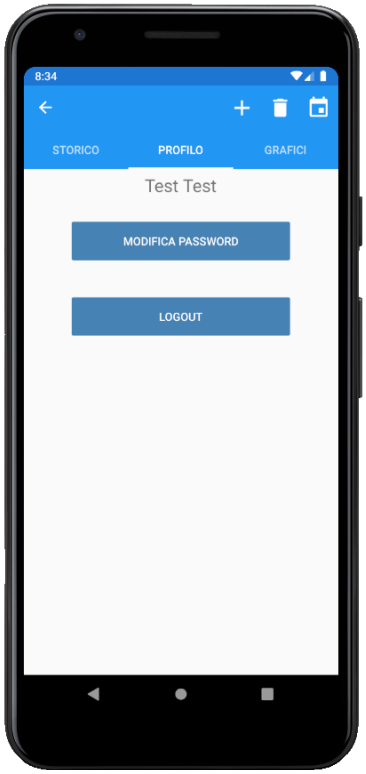
Login



Registrazione



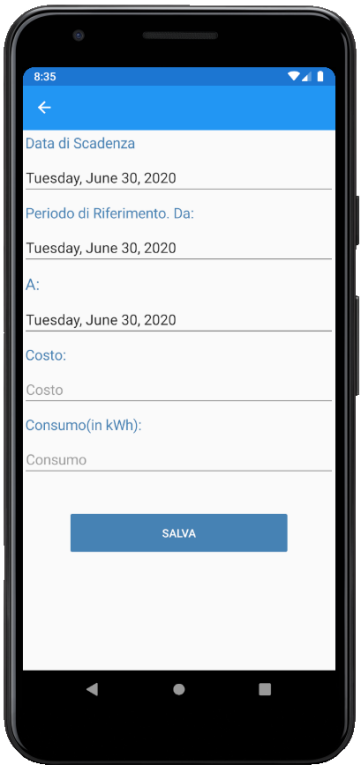
Storico Bollette



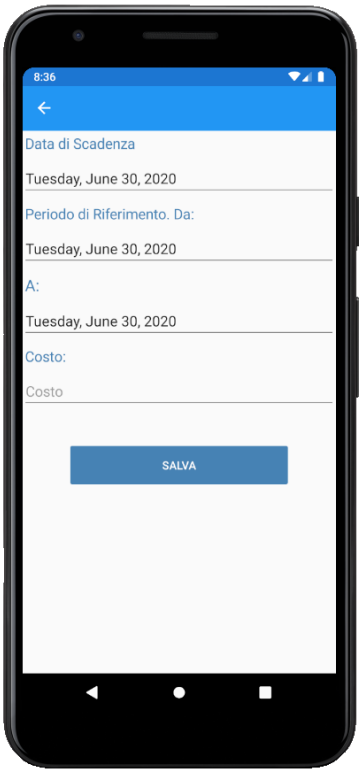
Profilo

Grafici

Selezione Bolletta



Form Inserimento Bolletta Luce/Gas



Form Inserimento Bolletta Internet

Considerazioni e differenze tra le due versioni

Abbiamo cercato di rendere il più possibile la versione Xamarin fedele a quella originaria Android. Ci sono state non poche difficoltà nell'implementazione di Firebase, perché mentre per Android e java c'è un'ampia e dettagliata documentazione ufficiale, per C# e Xamarin abbiamo sfruttato librerie GitHub di utenti, non esistendo documentazione e supporto di Firebase a riguardo.
In particolare, il plugin CloudFirestore(cross-platform) ci ha permesso di fare query al Database Firestore direttamente nello Shared Project, mentre per il login e la registrazione abbiamo usato i Dependency Services, metodi usati nello Shared Project che richiamano funzionalità native Android (Xamarin.Firebase.Auth non è cross-platform).
Una piccola differenza tra le due versioni è il fatto che in Android al click sul bottone del calendario viene aperta l'app Calendario scelta, in Xamarin l'aggiunta dell'evento viene fatta automaticamente e si rimane in BollettApp.
In generale abbiamo cercato di rendere le due versioni dal punto di vista grafico le più simili possibile, ci sono solo differenze stilistiche nei bottoni e nel posizionamento di qualche icona.
Per visualizzare lo storico bollette ci serviamo di due componenti differenti che, tuttavia, producono un risultato simile: in Android e Java usiamo una RecyclerView, in Xamarin e C# una CollectionView, quest'ultima caratterizzata da un binding automatico tra Model dei dati e Vista, fattore molto interessante e semplificativo in fase di progettazione.

Considerazioni finali

L’app sviluppata su Android nativo è molto più leggera di quella in Xamarin (20 MB contro 135 MB circa).
Le performance delle due applicazioni sono simili, entrambe riescono ad essere reattive ai tocchi e hanno dei caricamenti delle operazioni molto brevi. Sporadicamente si notano rallentamenti ad esempio nel login, ma crediamo siano dovuto a problemi di connessione non legati all’app.
Come gruppo, riteniamo che Xamarin sia rimasto un po’ indietro rispetto alle funzionalità introdotte da Android e IOS per permettere ai programmatori di sviluppare app complesse in modo più agevole.
Xamarin nasce per integrare funzionalità cross-platform, ma essendo sparito il sistema operativo di Windows dal mercato mobile, attualmente il suo potenziale e la sua utilità si sono ridotti, questo fa sì che si preferisca sviluppare direttamente applicazioni specifiche per Android e/o IOS sfruttando tutte le funzionalità specifiche dei due sistemi operativi.

Testing

Credenziali di accesso

Per mostrare tutto il potenziale di BollettApp abbiamo preparato degli utenti di prova con dati già inseriti, simulando quello che è il funzionamento dell'app nel corso di un anno intero e di un lungo periodo.

- Utente 1 **Email:** test@test.it ; **Password:** testtest
- Utente 2 **Email:** prova@prova.it ; **Password:** provaprova

Repository GitHub

La repository GitHub si riferisce alla versione Android nativa, per Xamarin abbiamo caricato su OneDrive uno zip per motivi di dimensione dei file. La repository contiene anche il relativo apk, mentre l'apk Xamarin è stato caricato su OneDrive.

Repository app Android nativa:
https://github.com/DanielePallini/BollettApp

Link OneDrive app Xamarin:
https://univpm-my.sharepoint.com/:u:/g/personal/s1083424_studenti_univpm_it/EVGNBdHu76xHou2pBvCjz9EBkKhpve-3MRpbN6lLa4NbGA?e=Ac6lKV

Link OneDrive apk Xamarin:
https://univpm-my.sharepoint.com/:u:/g/personal/s1083424_studenti_univpm_it/ESgjwsDbWmVNh6ohavYMX_cBOxGwh4T_d9dxzuOxZVeP5Q?e=X7Lyra

Nota:
Il progetto Xamarin è configurato in modalità release, per questo motivo se si prova a compilarlo si avrà un messaggio di errore, per risolverlo basta andare su Visual Studio in Compilazione->Gestione configurazione e cambiare le tre scelte a tendina che compaiono da "Release" a "Debug".