Università Politecnica delle Marche

Dipartimento di Ingegneria dell'Informazione

Realizzazione di un'App per la gestione delle spese di un fuorisede e l'organizzazione di tornei

Relazione per il corso di "Programmazione Mobile"
a cura dei docenti Domenico Ursino & Enrico Corradini

AA 2019/2020

Andreotti Vittorio
Bocci Matteo
Perticarari Samuele

Indice

Introduzione

Nell'ultimo decennio abbiamo assistito ad un aumento esponenziale dell'uso dei dispositivi mobile, utilizzati oggigiorno in ogni settore immaginabile.

L'utilizzo di questi dispositivi offre molti vantaggi sia al gestore di un determinato servizio sia al cliente o a chiunque utilizzi l'applicazione.

Il progetto, realizzato nell'ambito del corso di Programmazione Mobile, consiste nello sviluppo di un'applicazione per la gestione delle spese di uno studente fuorisede o non, e l'organizzazione di tornei tra i possessori dell'app.

Il progetto è stato realizzato in due versioni utilizzando due delle tre diverse tecniche di programmazione mobile: app nativa e app ibrida.

L'app nativa è stata realizzata per il sistema operativo Android, attraverso Android Studio come IDE e Java come linguaggio di programmazione; mentre l'app ibrida è stata sviluppata con il framework Xamarin, utilizzando Visual Studio come IDE e C# come linguaggio di programmazione.

Analisi dei requisiti

Descrizione del progetto

Il nostro progetto consiste nella realizzazione di un'App per la gestione delle spese di una casa dove risiedono uno o più studenti e l'organizzazione di tornei.

L'applicazione consente al primo accesso dell'utente di scegliere tra effettuare il Login o procedere con la registrazione, con la possibilità di registrarsi anche con Google.

Una volta effettuata la registrazione dovrà scegliere se creare un account relativo ad un proprietario di una casa oppure creare un affittuario. Gli step successivi differiscono in base alle scelte fatte precedentemente.

Per quanto riguarda il proprietario, esso creerà una nuova casa inserendo il nome della casa e l'indirizzo. L'utente affittuario invece andrà ad inserire un codice della casa per poter accedere a quella determinata casa.
Questo codice gli verrà fornito tramite la condivisione da parte del proprietario.

Nella homepage, in entrambi i casi ci sarà un grafico a torta che indica il riepilogo delle spese, con la divisione tra le spese pagate e non pagate e una successiva divisione interna per le tipologie di spese.

Nell'applicazione sarà presente un menu laterale dal quale è possibile eseguire varie operazioni. Gli utenti possono visualizzare il proprio profilo e hanno l'opportunità di modificare la password o di cancellare il proprio account.

Riguardo le spese, l'utente proprietario può inserire nuove spese (affitto, bollette e spese di condominio), mentre l'affittuario può inserire una spesa che verrà divisa con i coinquilini (spesa comune). Per entrambi è poi possibile visualizzare le spese, o nella scheda del sommario, dove compaiono tutte le spese sia pagate che non pagate, o nelle schede specifiche divise per tipologia della spesa.

In queste schede l'affittuario può marcare come pagate le spese, che poi risulteranno pagate nelle schede del proprietario.

I tornei invece prevedono una creazione di un nuovo torneo dove si specificherà: un titolo, una data, un orario, un indirizzo, la categoria del torneo e un regolamento; una visualizzazione di tutti i tornei dove possiamo confermare la partecipazione, e uno storico dove si visualizzano i tornei a cui l'utente ha partecipato. Nel menu laterale è poi possibile effettuare il Log out, che fa tornare l'interfaccia grafica alla schermata di login o registrazione.

Requisiti funzionali e non funzionali

In questa sezione vengono illustrati i requisiti funzionali e non funzionali dell'applicazione che si intende sviluppare. Rispettivamente, i requisiti funzionali rappresentano l'insieme delle funzionalità che dovranno essere implementate nell'applicazione; invece, I requisiti non funzionali, definiscono l'insieme dei vincoli e delle regole, sia di tipo realizzativo che tecnico.

Requisiti funzionali

Visto che l'applicazione che si intende sviluppare prevede l'accesso per due tipologie di utenti, è necessario dividere in due sezioni i requisiti funzionali e quelli non funzionali.

Requisiti funzionali condivisi

- Login con account Google.
- Login e registrazione.
- Inserimento di un torneo.
- Partecipazione ad un torneo.
- Lista tornei disponibili.
- Storico tornei ai quali si è partecipato.
- Profilo utente con possibilità di modificare la password.
- Log out.

Requisiti funzionali utente "Proprietario"

- Inserimento delle spese di vario genere per gli affittuari:
    - Inserimento spese d'affitto.
    - Inserimento bollette.
    - Inserimento spese condominiali.
- Visualizzazione di un riepilogo delle spese dell'intera casa (PAGATE e NON PAGATE) mediante un grafico.
- Visualizzazione di un sommario delle spese.
- Visualizzazione delle spese legate ad ogni singolo utente:
    - Visualizzazione spese d'affitto.
    - Visualizzazione bollette.
    - Visualizzazione spese condominio.

Requisiti funzionali utente "Affittuario"

- Inserimento delle spese comuni per i propri coinquilini
- Visualizzazione di un riepilogo delle spese del proprio utente (PAGATE e NON PAGATE) mediante un grafico.
- Visualizzazione delle spese:
    - Visualizzazione di un sommario delle spese.
    - Visualizzazione spese comuni.
    - Visualizzazione spese d'affitto.
    - Visualizzazione bollette.
    - Visualizzazione spese condominio.
- Marcare come pagata una spesa.

Requisiti non funzionali

Requisiti non funzionali condivisi

- In fase di registrazione l'utente deve inserire tutti i campi.
- Ogni casa deve avere un solo proprietario.
- Il codice della casa deve essere univoco.
- Ogni iscritto al servizio deve appartenere ad una sola tipologia d'utenza.
- Per inserire una spesa l'utente deve riempire ogni campo.
- Per creare un torneo l'utente deve specificare ogni campo.
- Gli utenti che hanno effettuato il Log-In utilizzando il proprio account Google non possono modificare la password.

Requisiti non funzionali utente "Proprietario"

- Gli utenti di tipo "Proprietario" non possono:
    - Inserire una spesa comune.
    - Eliminare il proprio account.
    - Segnare una spesa come "Pagata".

Requisiti non funzionali utente "Affittuario"

- Gli utenti di tipo "Affittuario" non possono:
    - Inserire spese appartenenti alle categorie "Bollette", "Affitto", "Spese condominio".

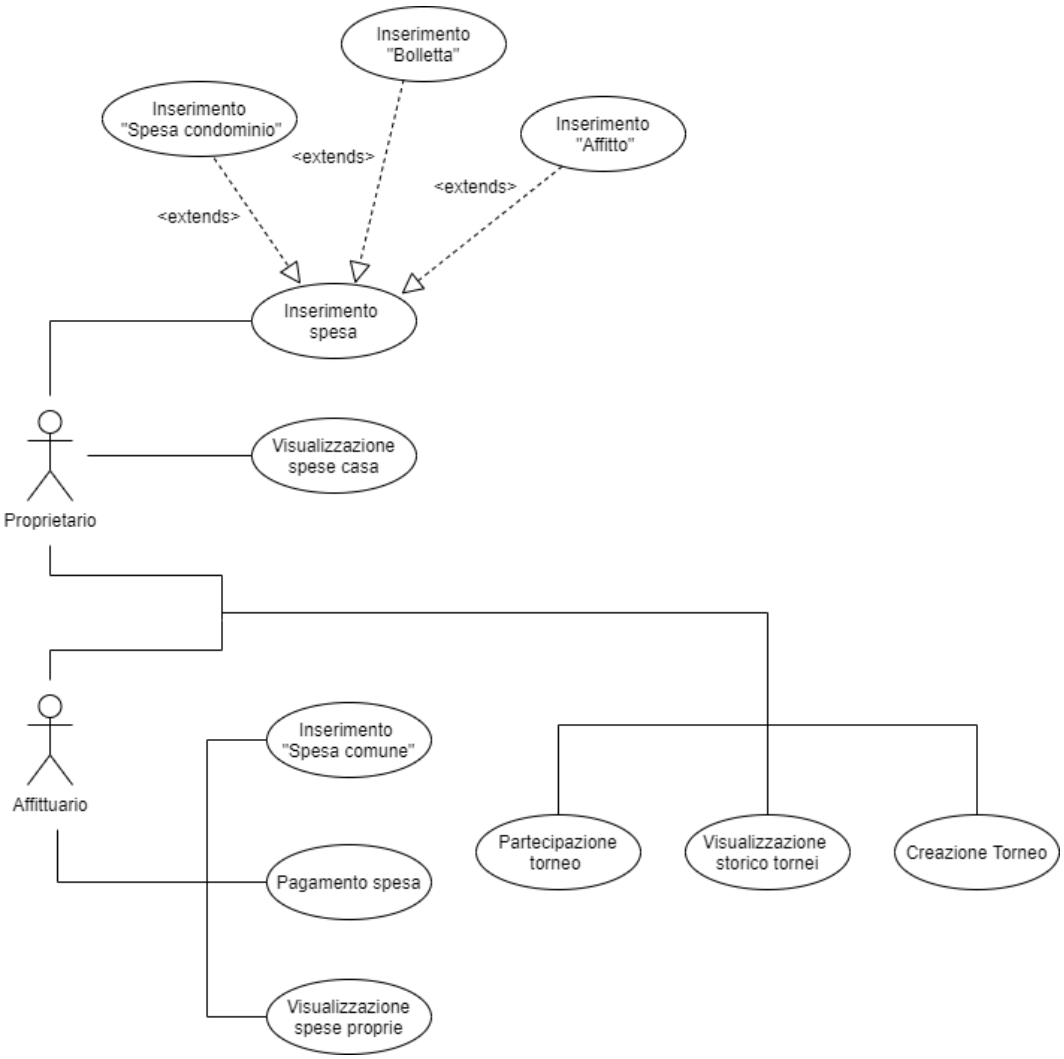
Attori e casi d'uso

I diagrammi dei casi d'uso sono dei diagrammi definiti in UML il cui scopo è quello di descrivere mediante rappresentazione grafica le funzionalità (casi d'uso) che un servizio mette a disposizione degli utenti (attori) che con esso interagiscono.

Nel nostro caso gli attori a cui sono offerte le funzionalità che l'applicazione mette a disposizione sono:

- Proprietario: utente identificato come proprietario a seguito della procedura di login. Dispone delle funzionalità di gestione delle spese (inserimento, visualizzazione di spese pagate e non pagate di tutti gli affittuari appartenenti alla casa di cui è proprietario) e gestione dei tornei (creazione, partecipazione e visualizzazione di tornei ai quali ha partecipato/parteciperà)
- Affittuario: utente identificato come affittuario a seguito della procedura di login. Dispone delle funzionalità di gestione delle spese (inserimento spesa comune, visualizzazione di spese pagate e non pagate) e gestione dei tornei (creazione, partecipazione e visualizzazione di tornei ai quali ha partecipato/parteciperà).

Diagramma dei casi d'uso



Progettazione

Progettazione della componente applicativa

Mappa dell'applicazione

La mappa dell'applicazione è una rappresentazione schematica della struttura dell'applicazione, che consente di visualizzare immediatamente come i contenuti forniti sono organizzati nelle diverse pagine, e quindi come l'utente può effettivamente accedervi.

L'organizzazione dei contenuti deve essere tale da permettere una navigazione intuitiva all'interno dell'app e quindi contribuire a garantire un'esperienza utente soddisfacente.

Al primo avvio dell'applicazione le strade che si prospettano davanti ad un utente del nostro servizio sono due: Login o Registrazione.

Riguardo il Login, l'utente può accedere alla propria area riservata inserendo le credenziali usate in fase di registrazione (eventualmente accedendo come utente Google).

Riguardo la Registrazione, l'utente può registrarsi inserendo i propri dati personali, scegliendo opportunamente E-mail e Password, oppure utilizzando il proprio account Google).
In una seconda fase viene presentata la schermata di scelta tra "Locatore" e "Locatario".

Se viene scelto "Locatore", viene presentata la schermata per poter creare una casa; invece, se si sceglie "Locatario", è necessario inserire il codice della casa per essere associati ad essa.

Il "Locatore", può condividere il codice della casa agli "Affittuari" per poterli far "partecipare" alla casa. Dopo aver partecipato alla casa o dopo averla creata (rispettivamente con un'utenza rispetto ad un'altra), viene aperta l'home dell'applicazione.

Nella home viene visualizzato un riepilogo sulle spese. Dal menù a lato è possibile accedere alle varie pagine dell'app. Il menù dell'affittuario differisce dal menù del proprietario, in quanto possono accedere a funzionalità differenti.

Mappa dell'applicazione – Sezione "Locatore"

Il "Locatore" nella home dell'applicazione può vedere un riepilogo delle spese della casa ad esso collegata.

Dal menù è possibile accedere alla sezione dell'inserimento delle spese.

Per quanto riguarda l'inserimento dell'affitto e della spesa condominiale l'importo è riferito ad ogni singolo utente, in quanto, solitamente sono spese che hanno una cadenza mensile oppure hanno un canone fisso per tutti gli affittuari.

Per quanto riguarda le bollette, l'importo della bolletta è diviso tra gli affittuari in egual misura.
Il locatore per condividere il codice della propria casa può utilizzare il pulsante presente nella pagina "Codice Casa" nella sezione "Casa".

Mappa dell'applicazione – Sezione "Locatario"

Il "Locatario" può partecipare alla casa di un "Locatore" se al momento dell'iscrizione inserisce il codice della casa, il quale è condiviso dal "Locatore".

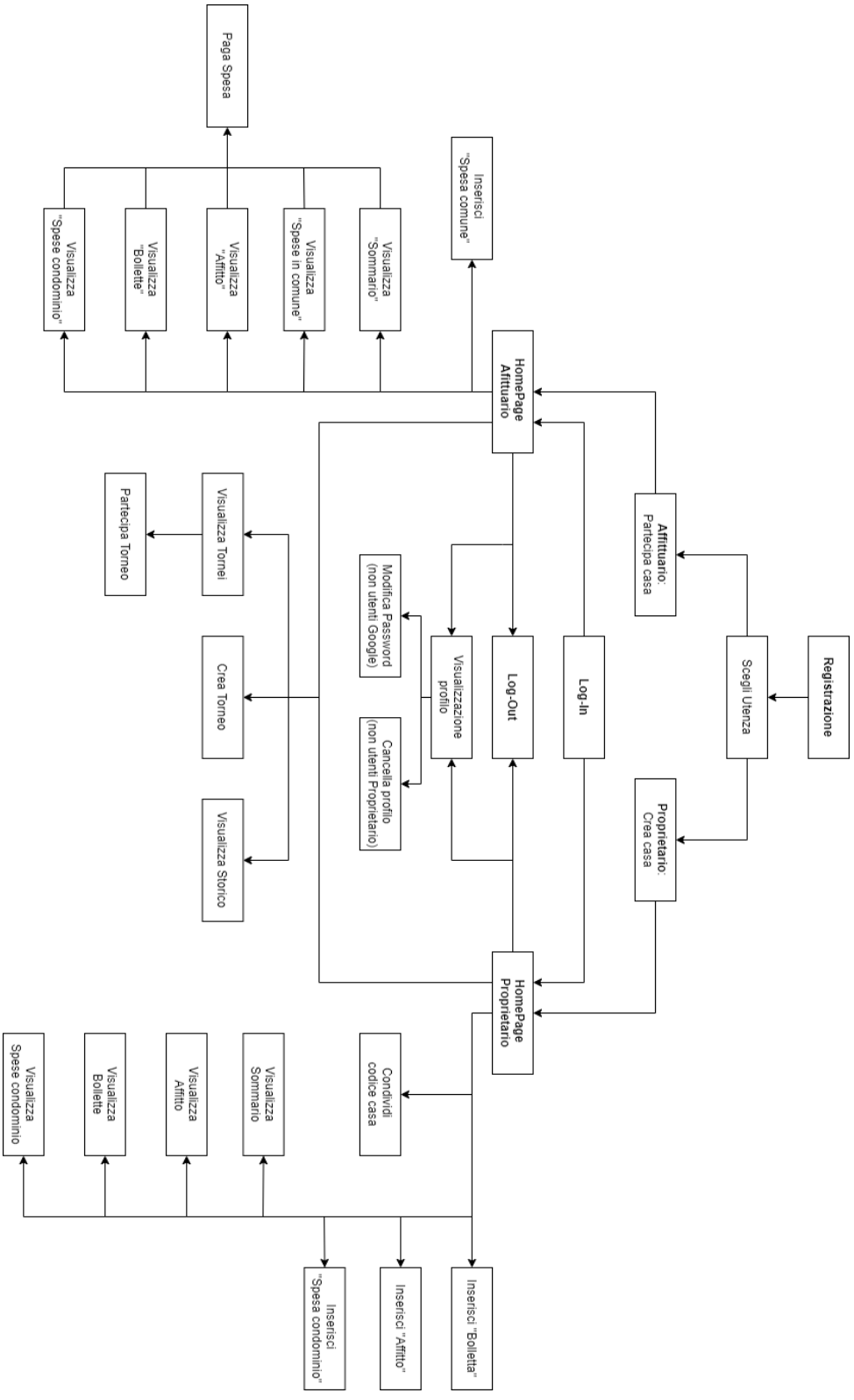
Il "Locatario" nella home dell'applicazione può vedere un riepilogo delle spese ad esso collegate.

Dal menù è possibile accedere alla sezione dell'inserimento delle spese comuni.

Le spese comuni sono le spese che fanno gli affittuari per la casa o per l'intero gruppo di coinquilini. Per esempio, è possibile considerare l'acquisto del detersivo come una spesa comune, oppure la compera di accessori per la casa.

Dal menù è possibile accedere alla sezione dell'inserimento delle spese comuni.

Le spese comuni sono le spese che fanno gli affittuari per la casa o per l'intero gruppo di coinquilini. Per esempio, è possibile considerare l'acquisto del detersivo come una spesa comune, oppure la compera di accessori per la casa.



Realizzazione dei Mockup

La realizzazione dei Mockup è un passo importante della progettazione dell'app, poiché ci permette di creare un'interfaccia grafica veloce per visionare a grandi linee il risultato finale. Esistono 2 livelli di mockup: livello 0 e 1.

Con mockup di livello 0 si intende la prima progettazione grafica, in questa app è stato realizzato con Balsamiq Mockups 3.

Con mockup di livello 1 invece si intende la rappresentazione grafica della schermata implementata.

Di seguito andremo a vedere i Mockup di livello 1 e 0 dell'applicazione creata in Android Studio, in particolare le seguenti schermate:

- Splash Screen (Figura 3).
- Scelta Login/Registrazione (Figura 4).
- Login (Figura 5).
- Registrazione (Figura 6).
- Scelta Proprietario/Affittuario (Figura 7).
- Creazione casa (Figura 8).
- Condivisione codice casa (Figura 9).
- Inserimento codice casa (Figura 10).
- Home page Affittuario – Proprietario (Figura 11 e Figura 12).
- Menu Affittuario – Proprietario (Figura 13 e Figura 14).
- Profilo (Figura 15).
- Creazione torneo (Figura 16).
- Partecipazione tornei (Figura 17).
- Storico tornei (Figura 18).
- Sommario spese Affittuario – Proprietario (Figura 19 e Figura 20).
- Affitto Affittuario – Proprietario (Figura 21 e Figura 22).
- Bollette Affittuario – Proprietario (Figura 23 e Figura 24).
- Spesa condominio Affittuario – Proprietario (Figura 25 e Figura 26).
- Spesa comune (Figura 27).
- Inserimento affitto (Figura 28).
- Inserimento bollette (Figura 29).
- Inserimento spesa comune (Figura 30).
- Inserimento spesa condominio (Figura 31).

(a) Mockup livello 0

(b) Mockup livello 1

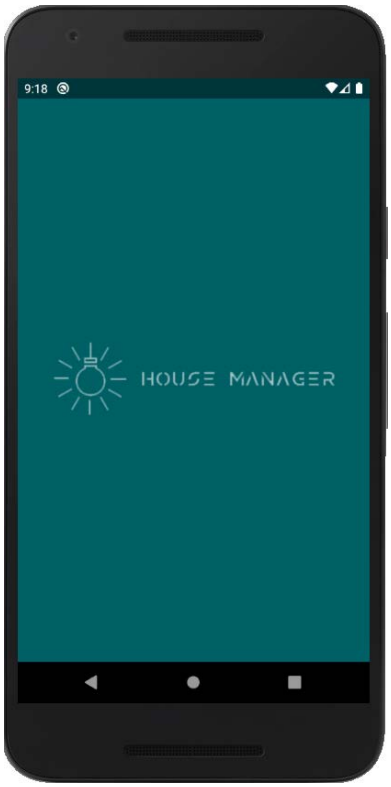
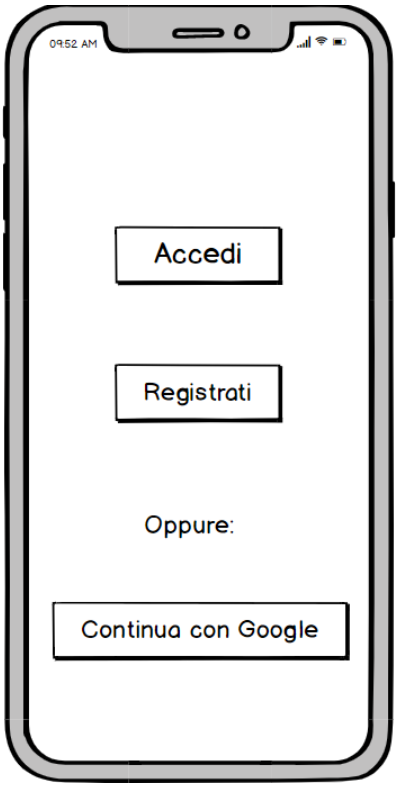


Figura 3: Mockup relativi alla splash screen iniziale dell'applicazione.



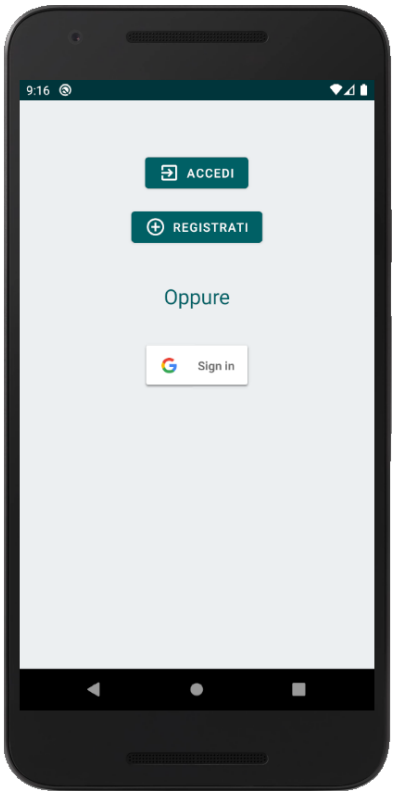


Figura 4: Mockup relativi alla scelta tra login e registrazione.

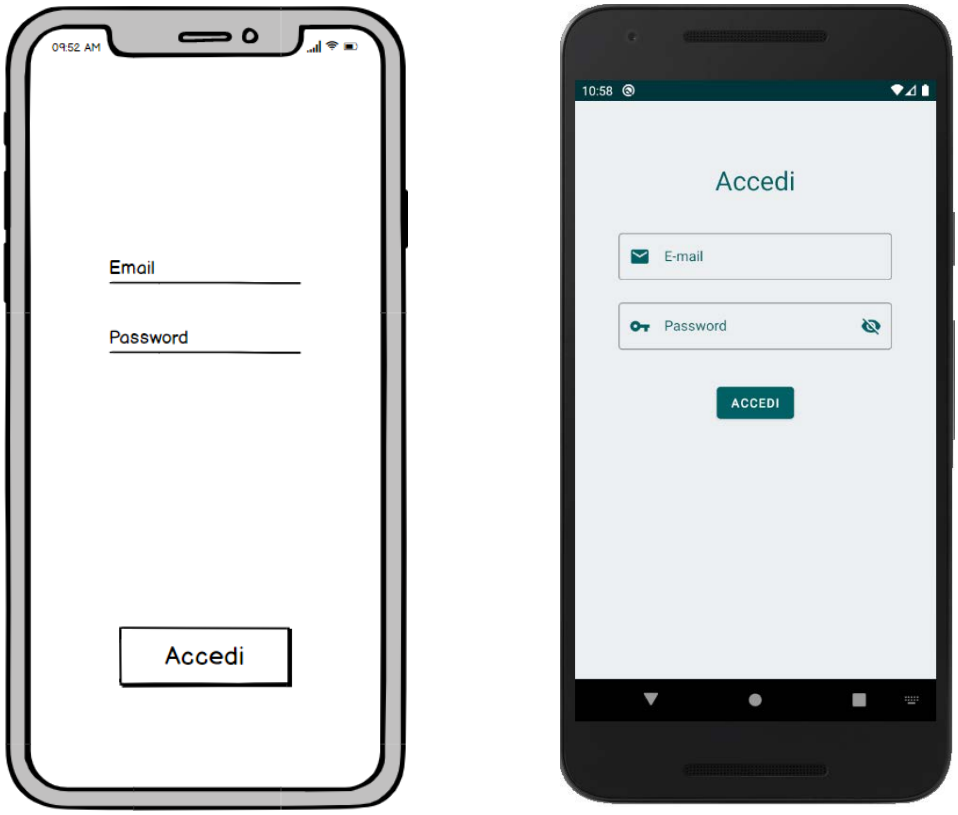


Figura 5: Mockup relativi alla pagina di login.

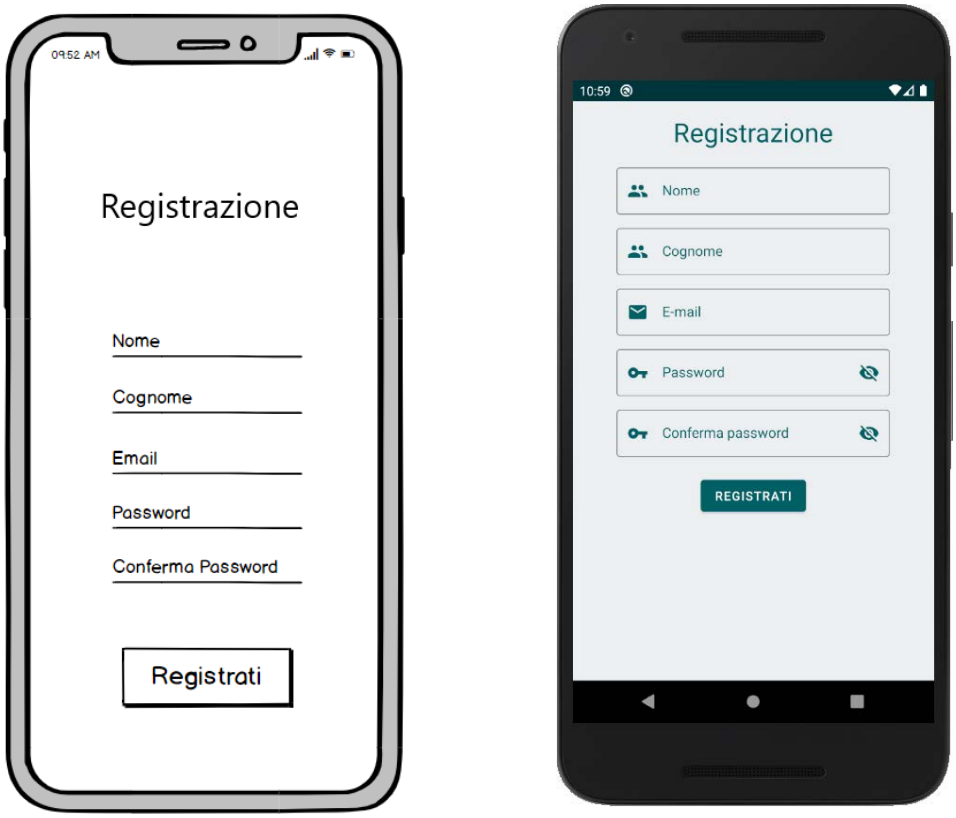


Figura 6: Mockup relativi alla pagina di registrazione.

Figura 7: Mockup relativi alla scelta tra proprietario e affittuario.

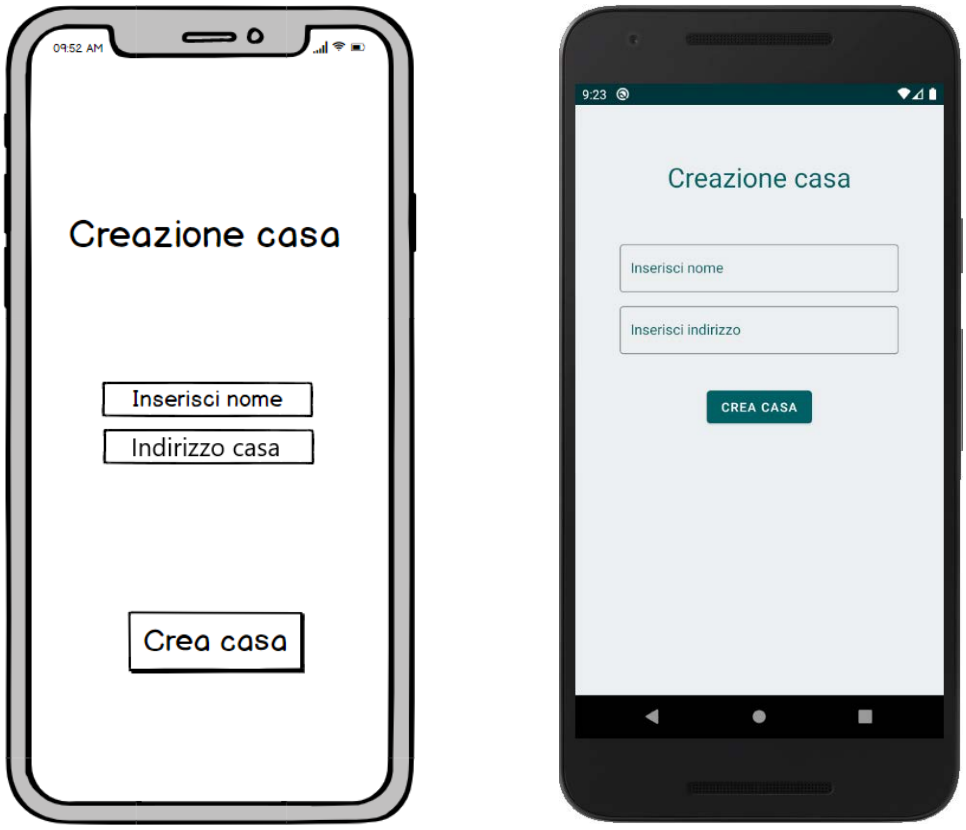


Figura 8: Mockup relativi alla pagina di creazione di una nuova casa.

Figura 9: Mockup relativi alla pagina di condivisione del codice della casa.

Figura 10: Mockup relativi alla pagina di partecipazione alla casa.

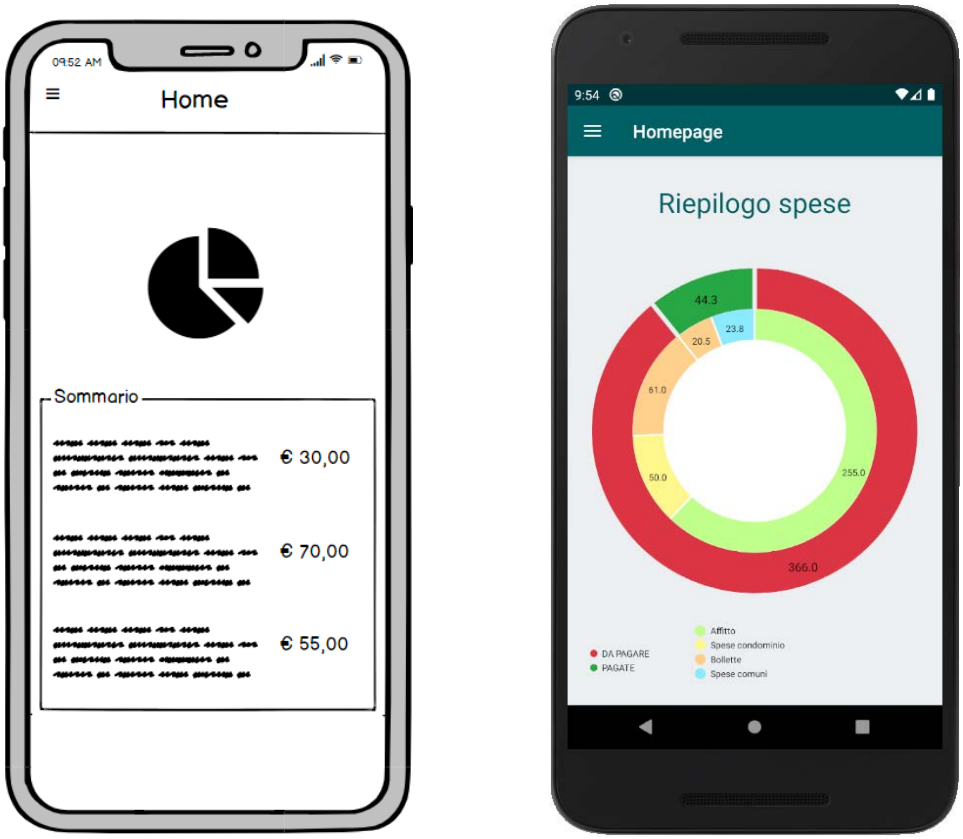


Figura 11: Mockup relativi alla homepage dell'affittuario.

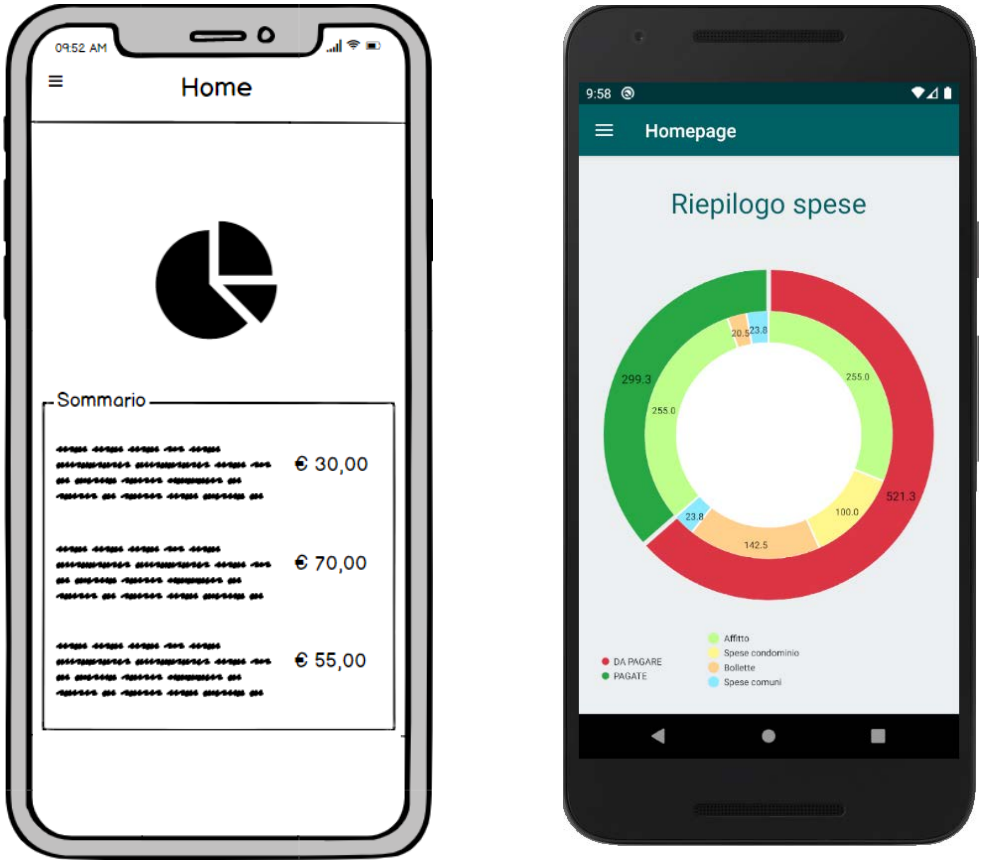


Figura 12: Mockup relativi alla homepage del proprietario.

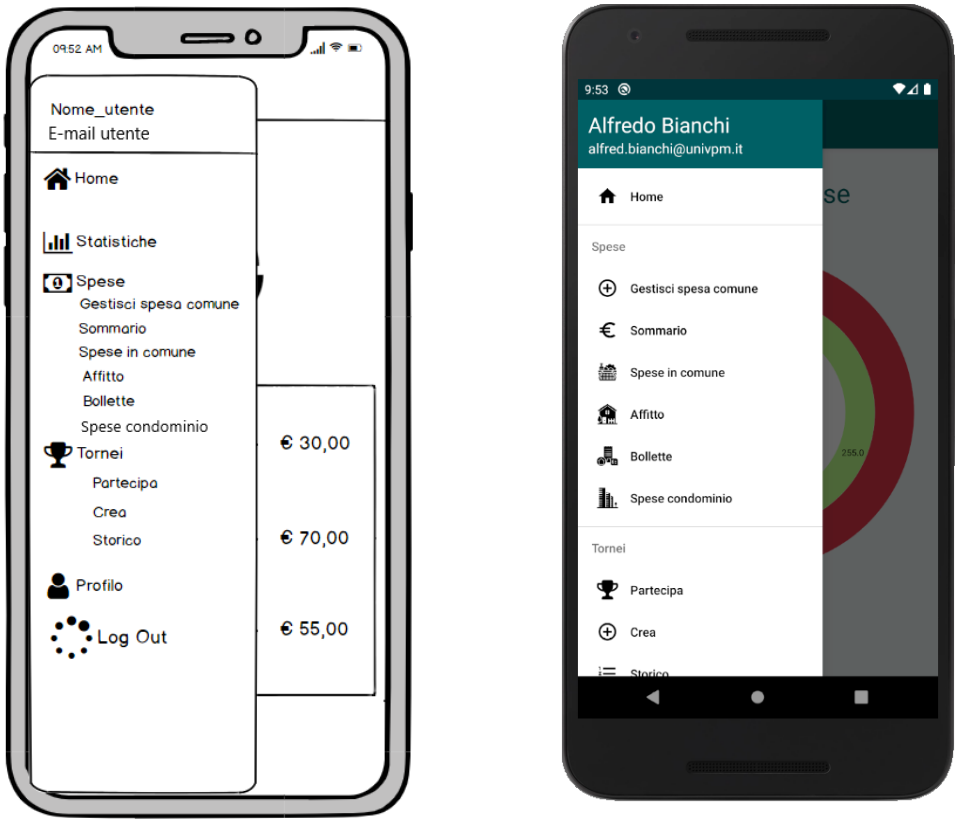


Figura 13: Mockup relativi al menu dell'affittuario.

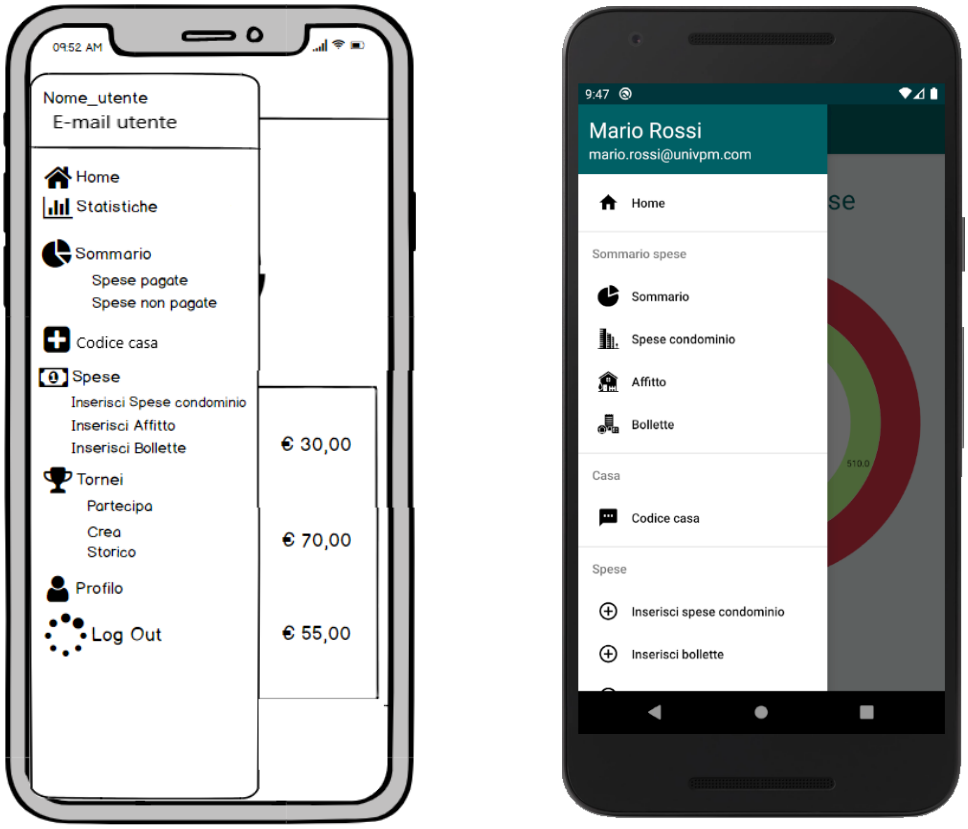


Figura 14: Mockup relativi al menu del proprietario.

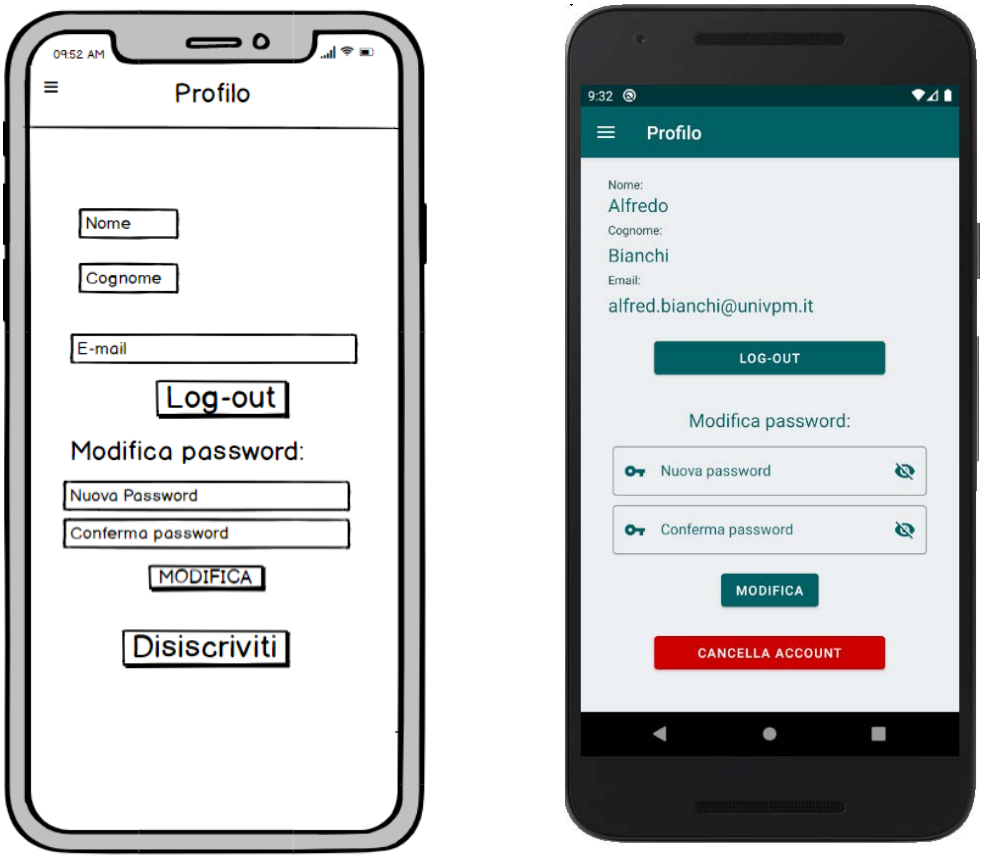


Figura 15: Mockup relativi alla pagina del profilo.

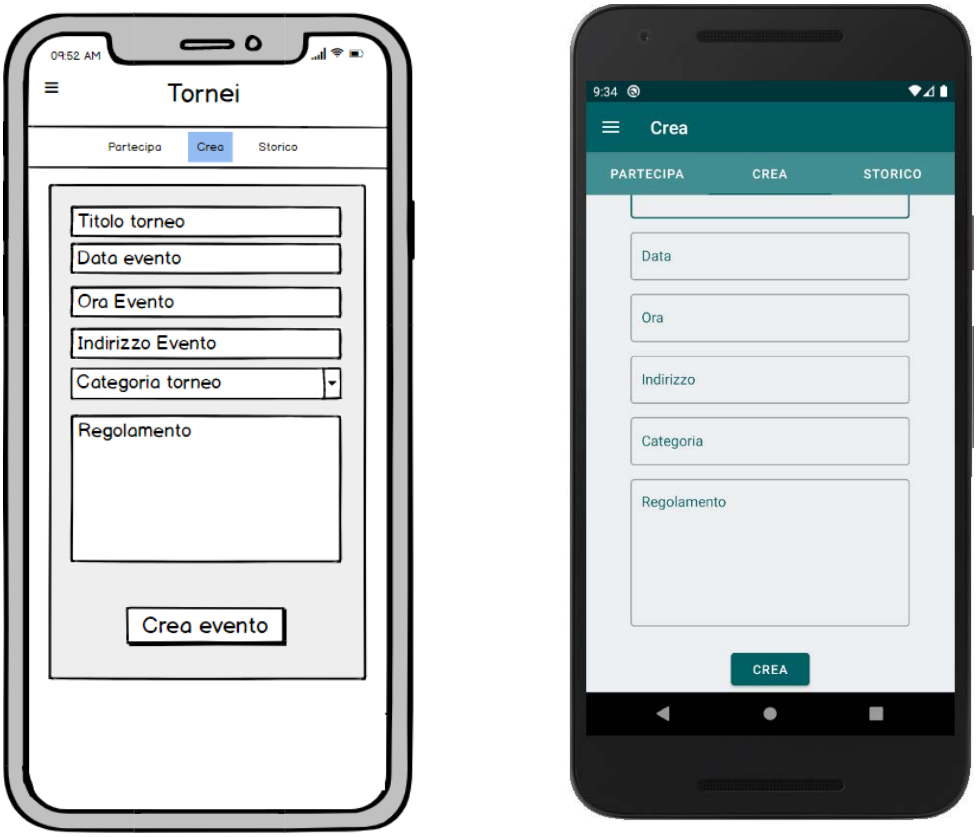


Figura 16: Mockup relativi alla pagina della creazione di un torneo.

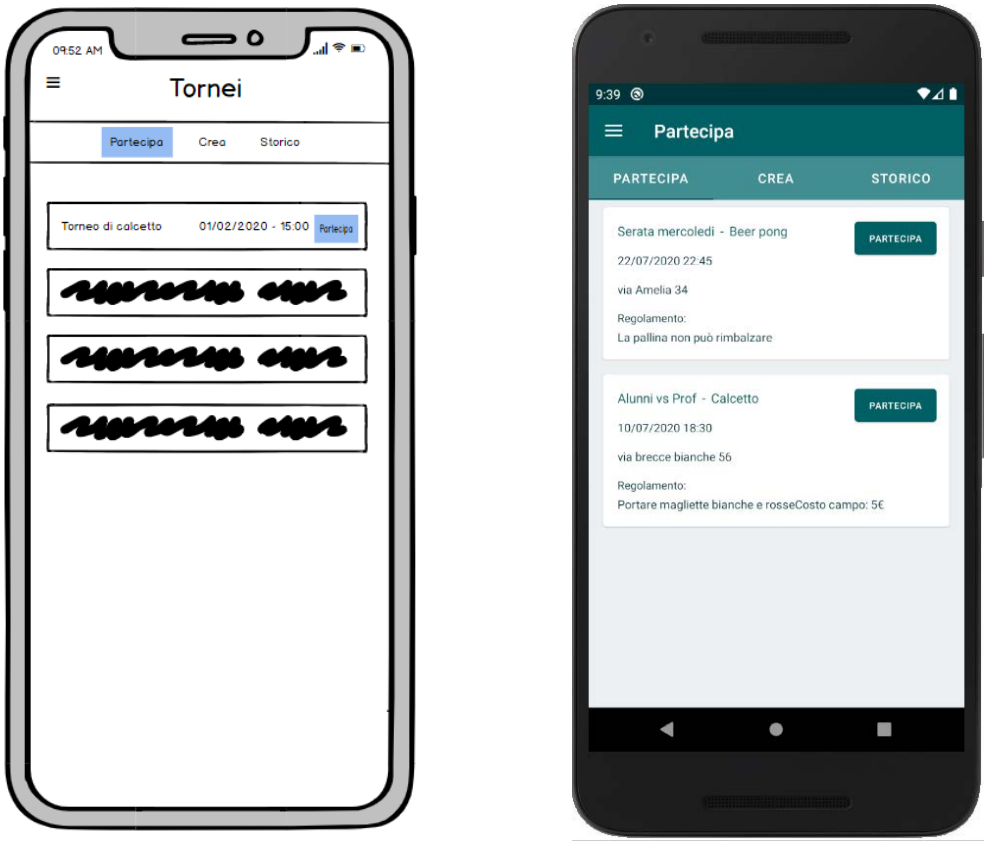


Figura 17: Mockup relativi alla pagina di partecipazione ad un torneo.

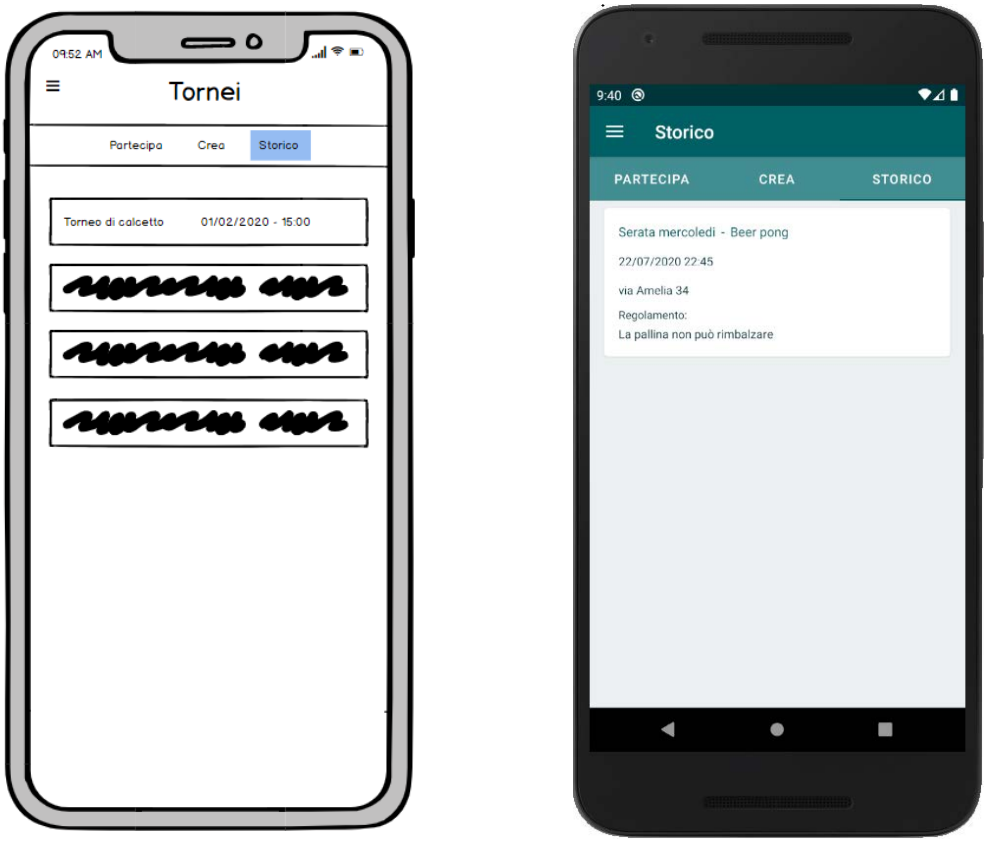


Figura 18: Mockup relativi alla pagina dello storico delle partecipazioni ai tornei.

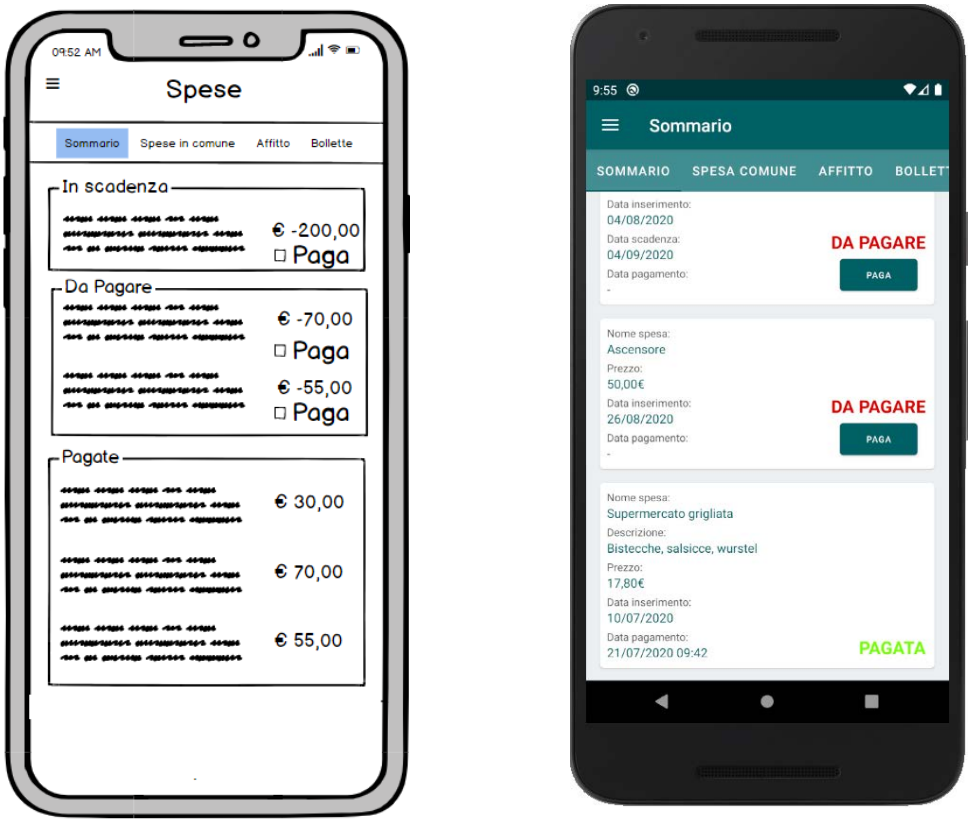


Figura 19: Mockup relativi al sommario delle spese dell'affittuario.

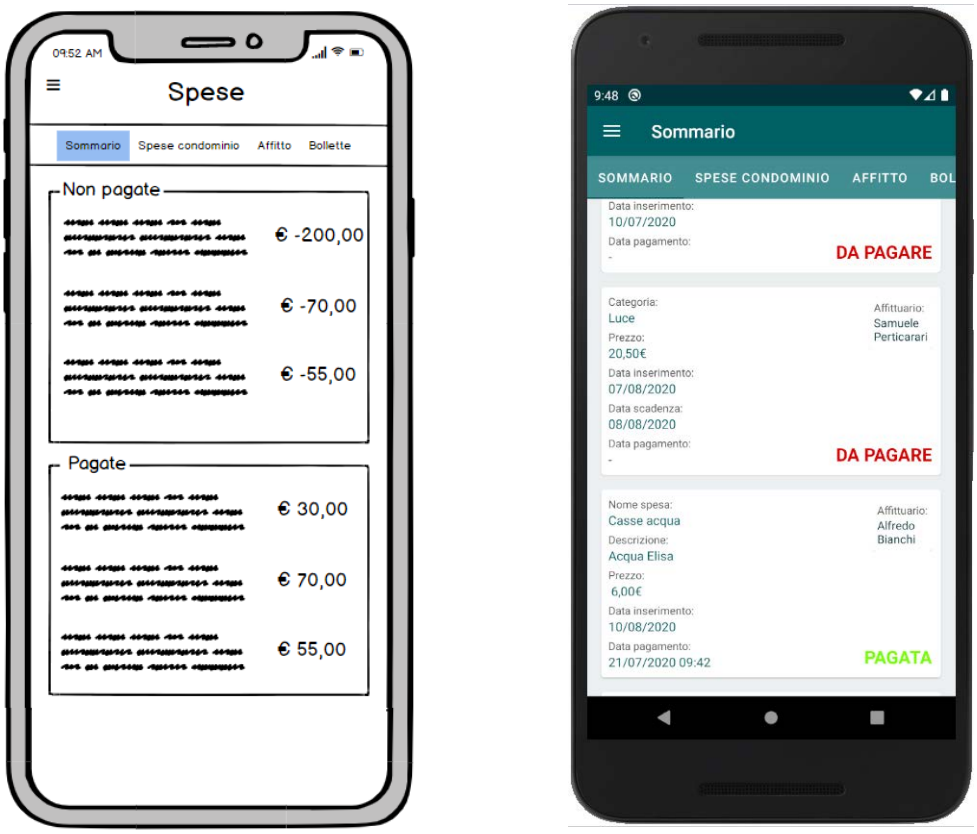


Figura 20: Mockup relativi al sommario delle spese del proprietario.

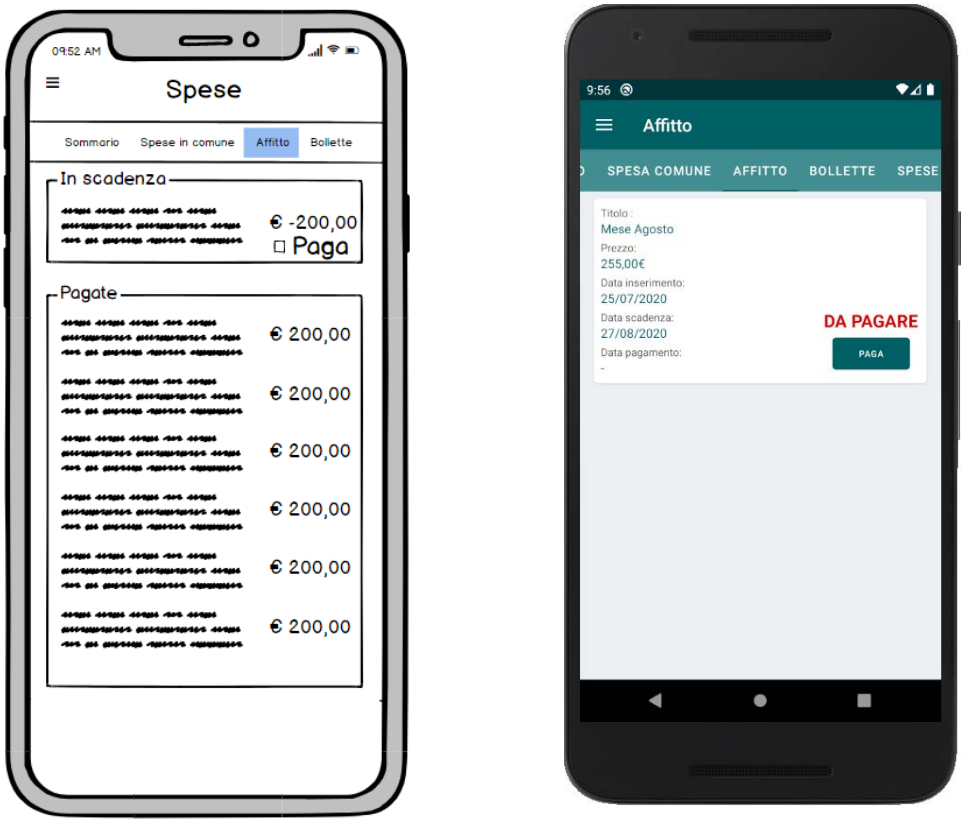


Figura 21: Mockup relativi alla scheda dell'affitto dell'affittuario.

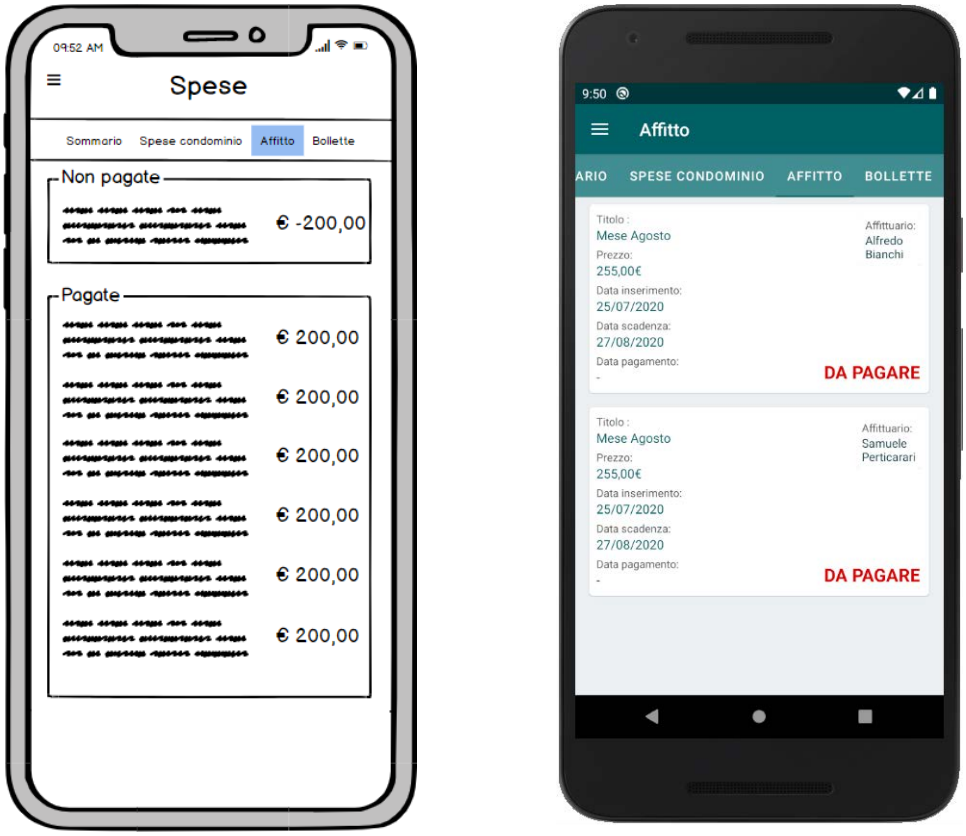


Figura 22: Mockup relativi alla scheda dell'affitto del proprietario.

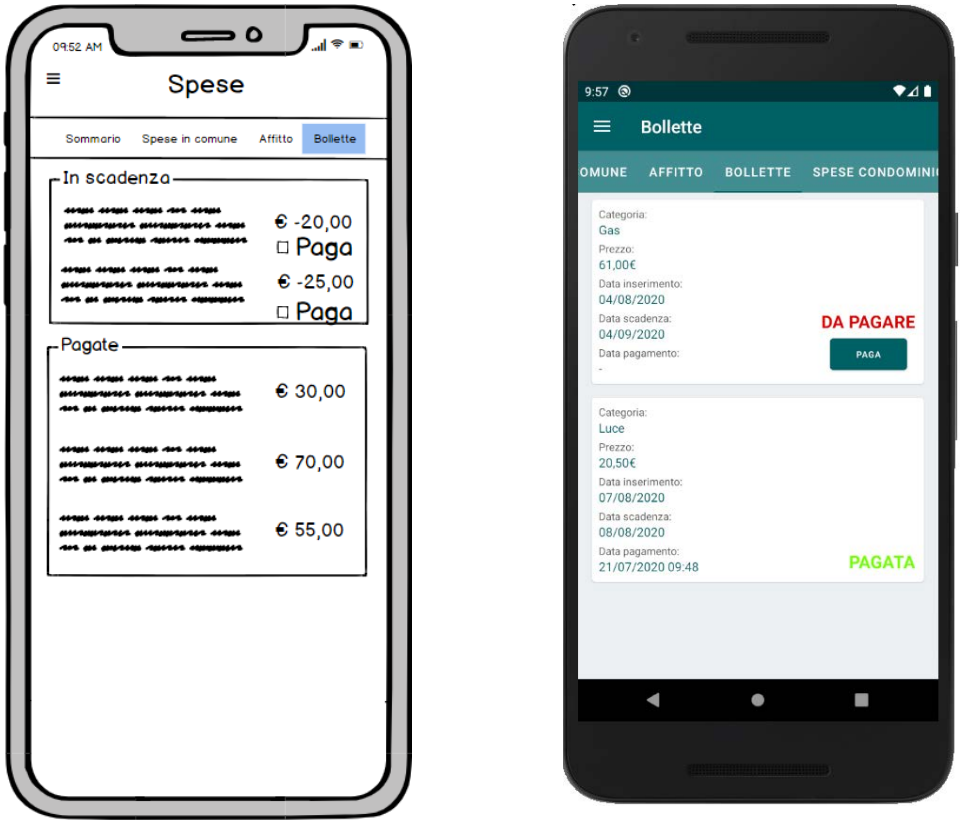


Figura 23: Mockup relativi alla scheda delle bollette dell'affittuario.

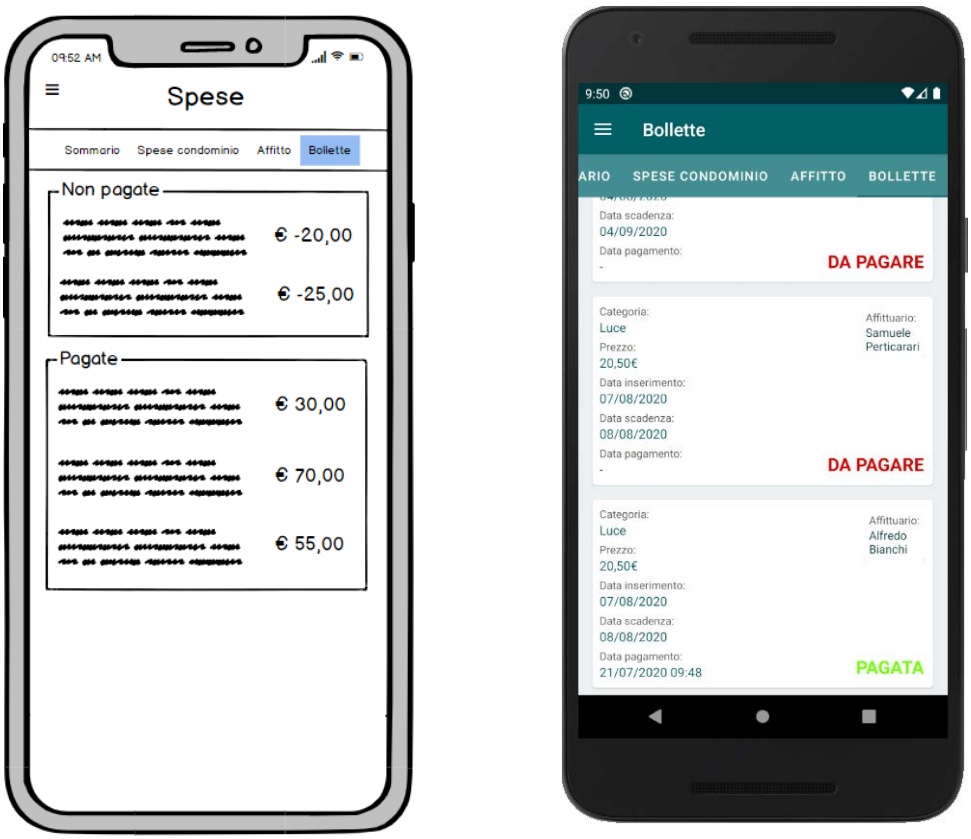


Figura 24: Mockup relativi alla scheda delle bollette del proprietario.

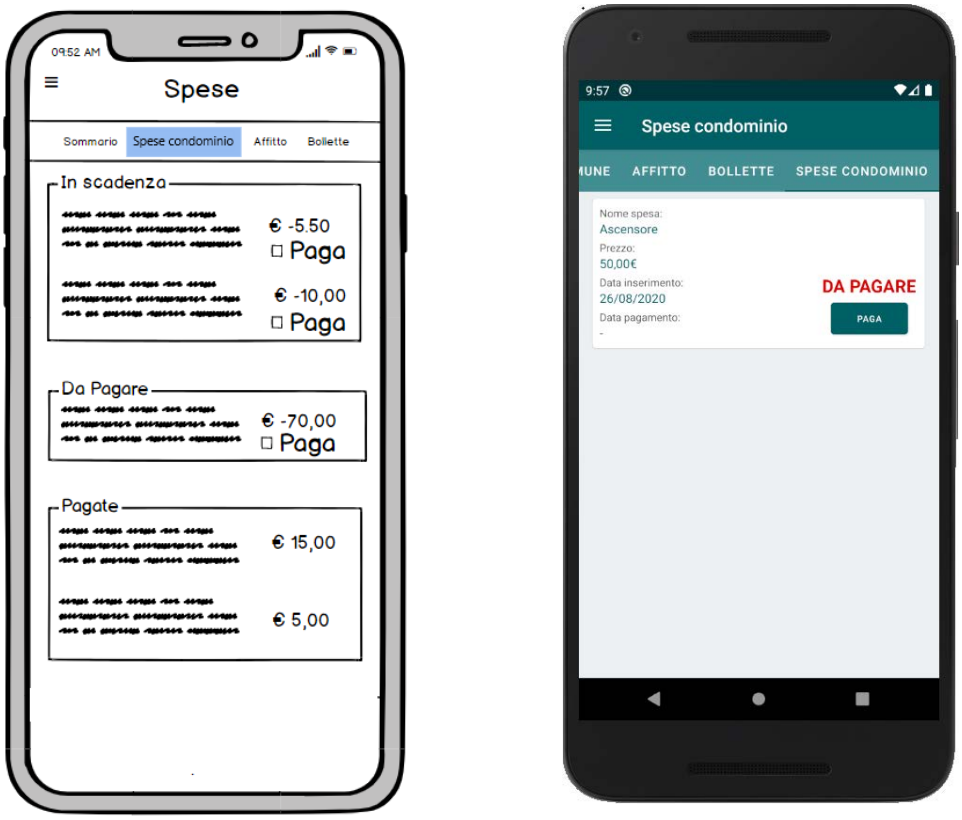


Figura 25: Mockup relativi alla scheda delle spese condominiali dell'affittuario.

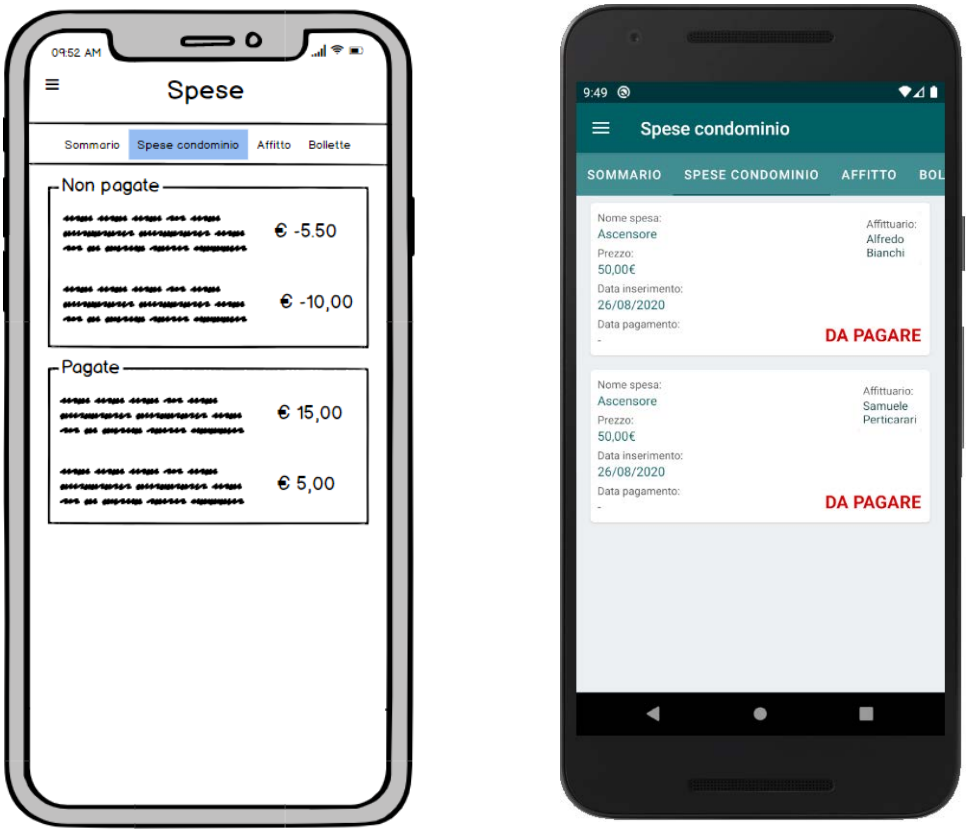


Figura 26: Mockup relativi alla scheda delle spese condominiali del proprietario.

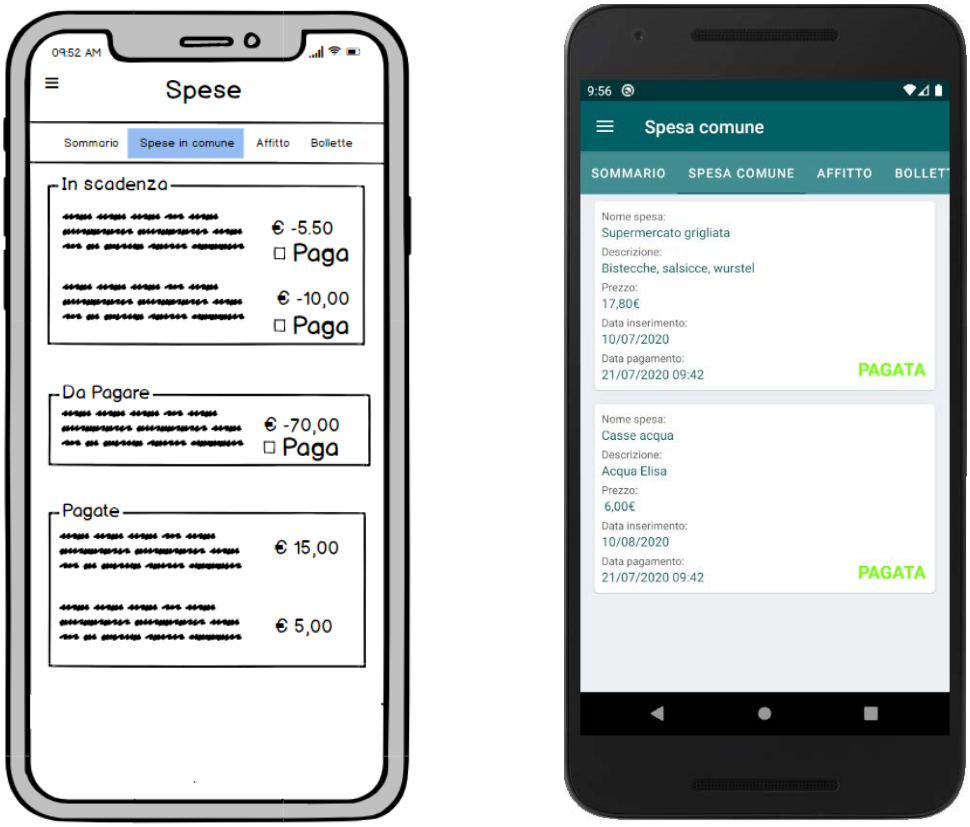


Figura 27: Mockup relativi alla scheda delle spese comuni.

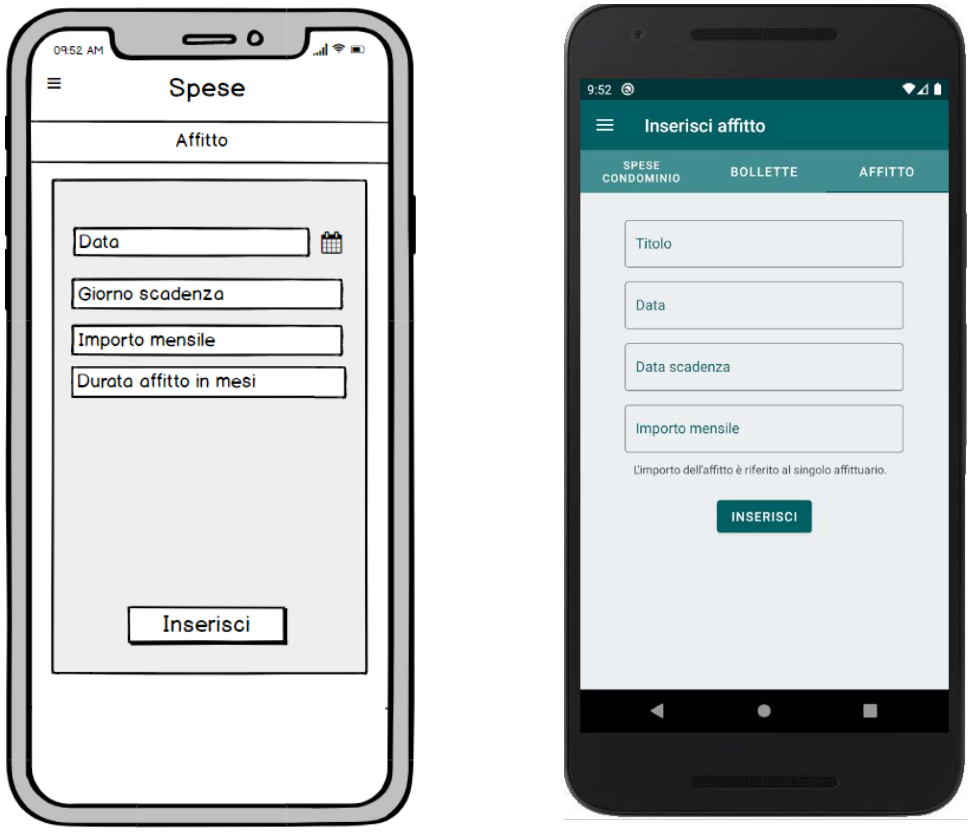


Figura 28: Mockup relativi alla scheda dell'inserimento dell'affitto.

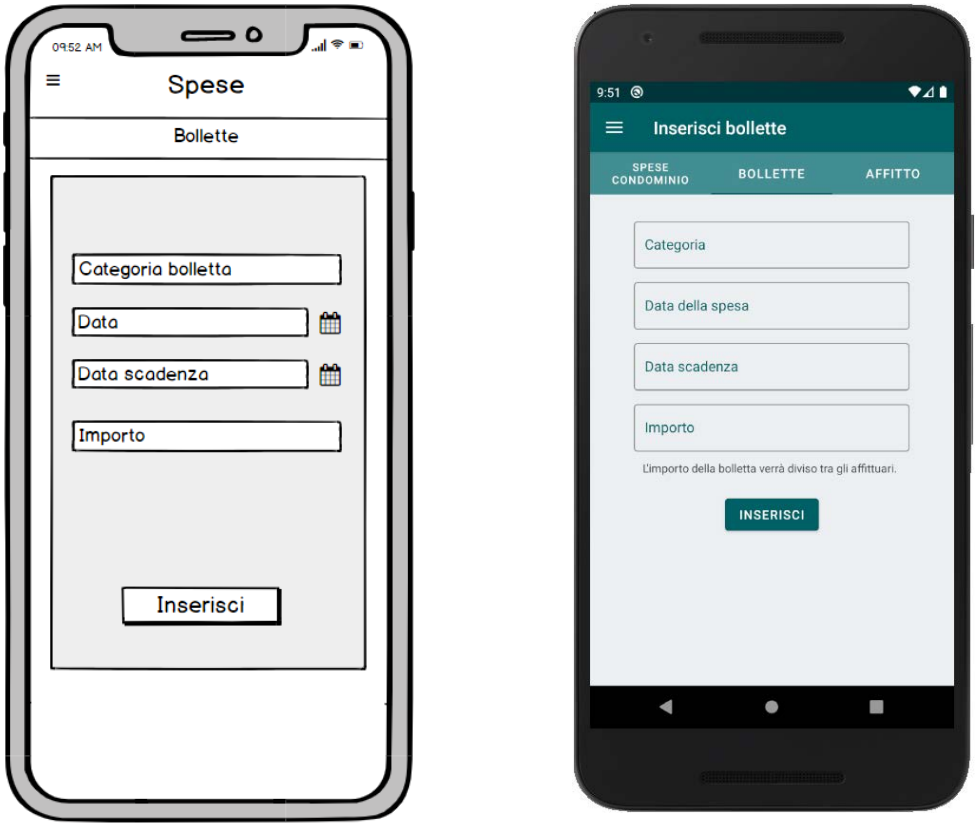


Figura 29: Mockup relativi alla scheda dell'inserimento delle bollette.

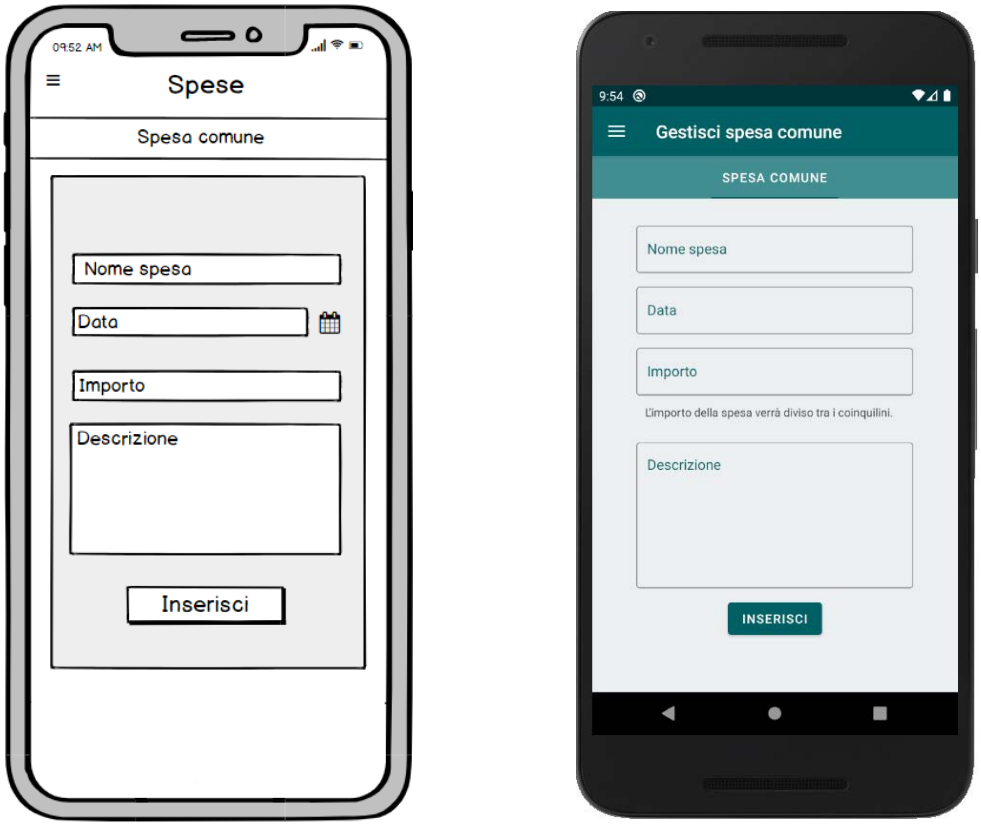


Figura 30: Mockup relativi alla scheda dell'inserimento della spesa comune.

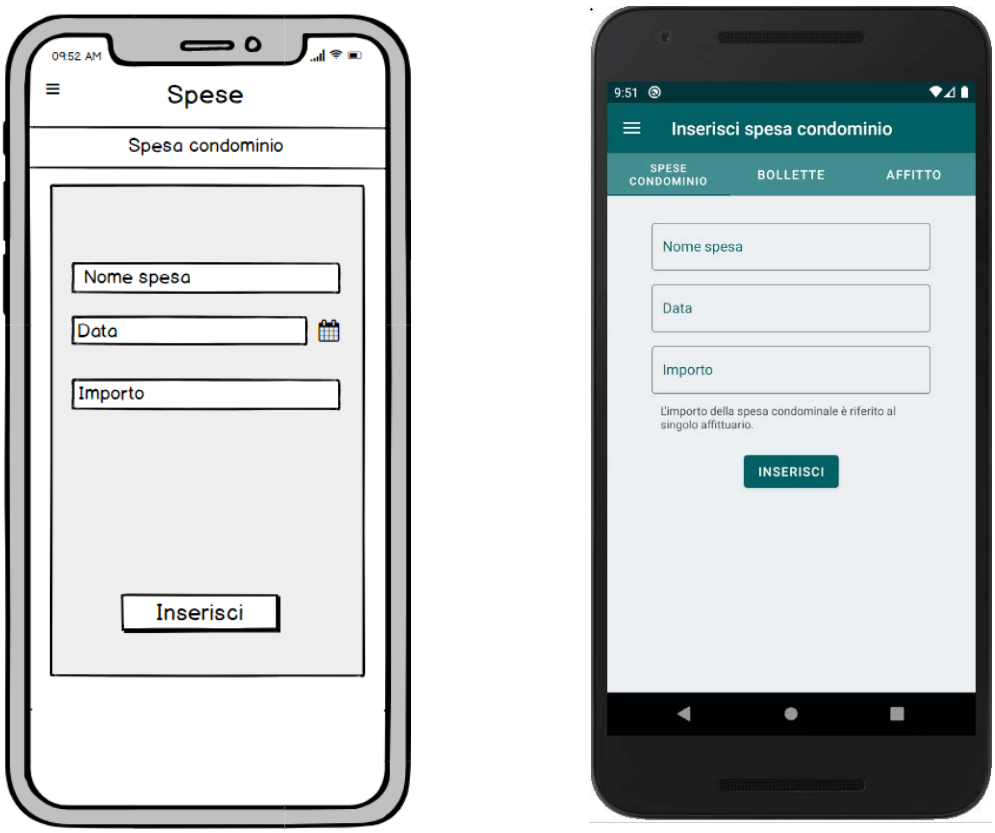


Figura 31: Mockup relativi alla scheda dell'inserimento della spesa di condominio.

Implementazione backend

Firebase Cloud Functions

Come backend abbiamo deciso di utilizzare le Cloud Functions di Firebase che si interfacciano con un database Firestore.

Le Cloud Functions per Firebase sono un *framework serverless* che consente di eseguire automaticamente il codice backend in risposta agli eventi attivati dalle funzionalità di Firebase e dalle richieste HTTPS.

Il codice JavaScript (o TypeScript) è archiviato nel *Cloud* di Google e viene eseguito in un ambiente gestito.
Trattandosi di una tecnologia serverless non è necessario gestire il server in quanto Google lo gestisce in automatico; quindi non è necessario effettuare nessun setup di esso, potendosi concentrare sul codice del 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 e largamente supportato da Android, IOS e dalle pagine Web.

Considerazioni sul database

Firebase ha due opzioni di database: Cloud Firestore e Realtime Database.
Entrambi sono database non relazionali NoSQL.

Un database NoSQL (Not Only SQL) non conserva i dati in tabelle definite a priori ma utilizza una struttura basata su raccolte di documenti Json, ciascuno composto da un id e da una serie di campi.

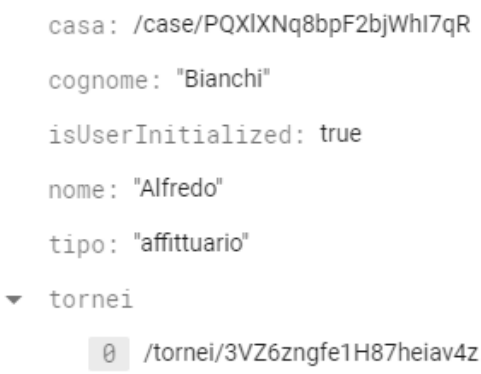
Per questo, il database manca di una strutturazione rigida dei contenuti; questo però non significa che vi sia assenza completa di regole, ma che la struttura dei documenti è dipendente dall'implementazione sul client per essere corretta.

Ciò ha due effetti significativi: per enormi quantità di dati, si hanno performance migliori rispetto ad un database SQL che deve verificare la correttezza dei dati (integrità dei dati) e quella delle relazioni fra tabelle (integrità referenziale), oltre che avere una maggiore scalabilità.

Implementazione database

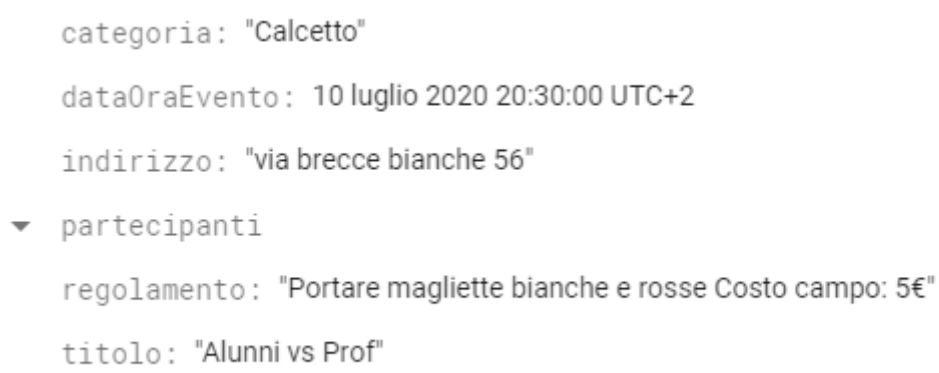
Raccolta *Utenti*

users (nome: string, cognome: string, tipo: string, isUserInitialized: boolean, casa: reference, tornei: reference)



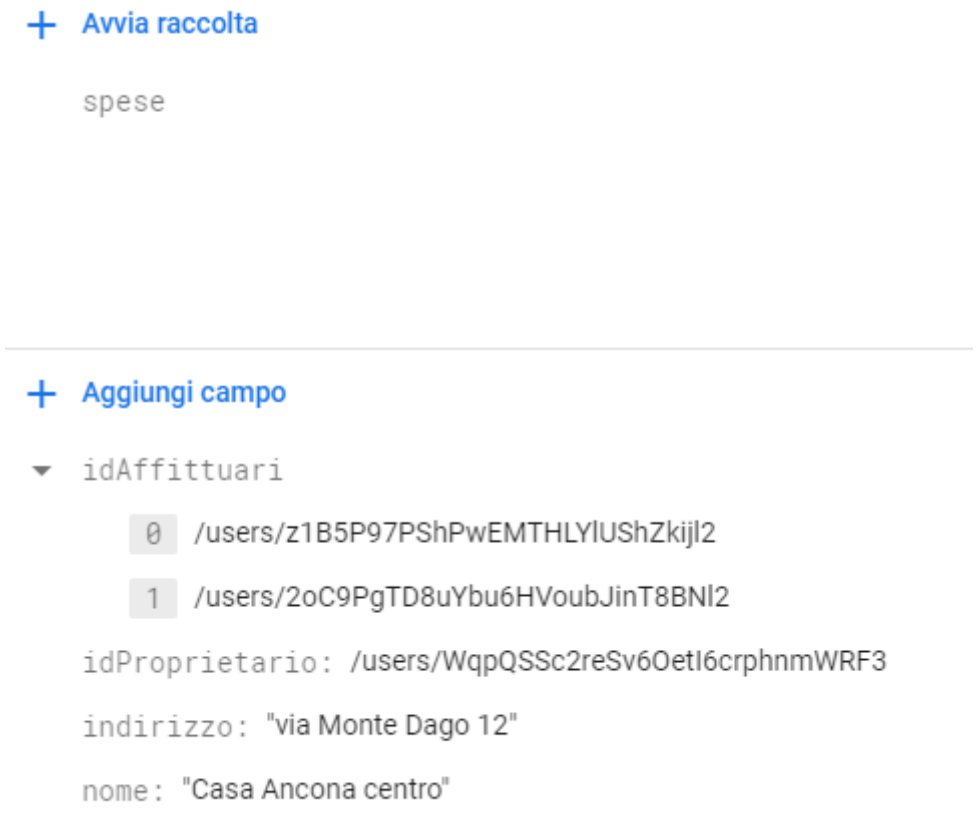
Raccolta *Tornei*

tornei (categoria: string, indirizzo: string, regolamento: string, titolo: string, dataOraEvento: timestamp, partecipanti: array<reference>)



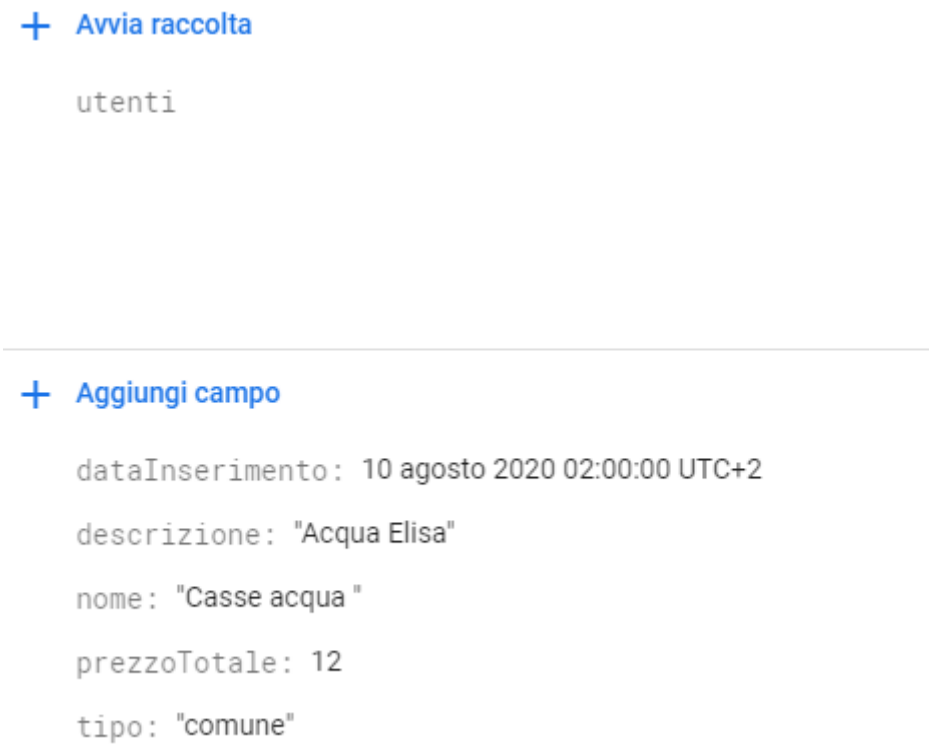
Raccolta *Case*

case (nome: string, indirizzo: string, idAffittuari: array<reference>, idProprietario: reference)



Raccolta Spese (nel singolo documento in Case)

spese (titolo: string, tipo: string, dataInserimento: timestamp, dataScadenza: timestamp)



Raccolta Utenti (nel singolo documento in Spese)

utenti (prezzo: number, dataPagamento: timestamp)

Implementazione dell'app – Android

L'applicazione è stata sviluppata utilizzando due tecniche differenti:

- Per l'app nativa in **Android** l'ambiente di sviluppo integrato (IDE) utilizzato è **Android Studio**, mentre il linguaggio di programmazione scelto è il **Java.**
- Per l'app **cross-platform**, in grado di essere eseguita su differenti sistemi operativi, l'**IDE** utilizzato è **Visual Studio**, basando lo sviluppo sull'utilizzo della funzionalità **Xamarin.Forms** del Framework **Xamarin** e del linguaggio **C#.**

Nel particolare, è possibile suddividere concettualmente l'applicazione secondo le seguenti macro-categorie:

- ***Login, Registrazione, Log Out*** : permette all'utente di accedere ad una serie di funzionalità diverse per la gestione del proprio account, riguardanti la possibilità di potersi registrare ed effettuare il login (sfruttando la propria E-mail oppure il proprio account Google), il log out dall'area riservata e la cancellazione dello stesso account (solo per utenti "Affittuario").
- ***Spese :*** permette di gestire le spese (inserire, visualizzare e contrassegnarle come pagate) appartenenti a diverse tipologie. Le varie funzionalità messe a disposizione sono differenziate in base alla tipologia d'utenza.
- ***Tornei :*** permette di gestire i tornei (creazione, visualizzazione di uno storico, visualizzazione di quelli ai quali non si ha partecipato, partecipazione). Le funzionalità sono identiche per entrambe le tipologie d'utenza.

NOTA:

La scelta di dare all'utente che si registra come Proprietario la possibilità di creare una casa è legata alla sola necessità di attuare un "collegamento logico" tra ogni utente Affittuario ed una **Casa**, in modo che le spese potessero essere gestite in modo agevole sia per il Proprietario che per l'Affittuario.

Per tal motivo all'interno dell'applicazione non esiste alcun altro riferimento ad essa, se non quella relativa al Proprietario, che ha la possibilità di condividere il proprio codice in modo che i suoi Affittuari possano partecipare.

Consapevoli che avremmo potuto adottare scelte differenti, questa strada ci è sembrata quella che rispecchiasse nel modo più fedele la realtà da noi considerata.

Gestione Login e Log Out

L'app è stata sviluppata in modo tale che al primo avvio venga mostrata la schermata di login o registrazione che permette all'utente di effettuare l'accesso con le proprie credenziali o di registrarsi e quindi di accedere alla propria homepage.

Utilizzando un'istanza di FirebaseAuth è possibile controllare se l'utente ha effettuato precedentemente il login per poter aprire direttamente la Homepage.

La procedura di log out consiste nella cancellazione delle shared preferences e nella chiamata al metodo signOut di un'istanza di FirebaseAuth.

Gestione lista spese e tornei

Nell'applicazione sono presenti diverse liste:

- lista spese nella homepage dell'affittuario (sommario, affitto, bollette, spese condominio e spese comuni) suddivise in pagate e da pagare, con i vari attributi in base alla tipologia ed eventualmente il bottone per pagare;
- lista spese nella homepage del proprietario (sommario, affitto, bollette, spese condominio) suddivise in pagate e non pagate, con i vari attributi in base alla tipologia e il nome e cognome dell'affittuario a cui è riferita;
- lista tornei e tornei a cui si ha partecipato(storico) con i vari attributi quale titolo, indirizzo, data, ora, categoria e regolamento.

Le varie liste sono gestite in modo analogo pur utilizzando due diversi layout e *ListAdapter*.
Gli adapter sono infatti degli *ArrayAdapter* che popolano le liste utilizzando i dati contenuti in vettori di stringhe opportunamente popolati dai task asincroni.

Drawer menu e toolbar

L'*HomeActivity* è l'activity che viene lanciata una volta che è stata verificata l'identità dell'utente autenticato nell'app. In questa activity è stato creato il Drawer Menu e una Toolbar (un classico nell'architettura di sviluppo delle applicazioni Android).
Dalla toolbar è possibile aprire il drawer menu cliccando sul pulsante in alto a sinistra.

Il Drawer Menu è inizializzato in maniera asincrona appena viene lanciata l'Activity in quanto al momento della prima registrazione o dopo aver effettato il login è necessario scaricare le informazioni sull'utente.

Una volta lette le informazioni sull'utente viene inizializzato il menu conseguentemente al tipo di utente che ha effettuato il login. Viene quindi caricato il menù e associata ad ogni elemento un azione che permette di caricare il fragment di interesse nell'elemento placeholder presente nella vista.
Ogni volta che è caricato un fragment viene settato di conseguenza il titolo della Toolbar.

Connessione alle Cloud Functions

L'applicazione utilizza un database esterno, necessario per gestire i dati (spese, tornei e utenti).

Per ottenere dati coerenti con il dato nel database Firestore abbiamo deciso di non storicizzare in locale le spese e i tornei relativi all'utente corrente, in quanto sono dati che vengono aggiornati molto spesso; sarebbe quindi inutile salvarli in locale, perché la copia in locale avrebbe dati non coerenti con i dati nel database, necessitando di un aggiornamento.

Di conseguenza la maggior parte delle azioni che l'utente può compiere all'interno dell'applicazione comporta una chiamata alle Cloud Functions attraverso la rete. Abbiamo migliorato l'user experience inserendo una schermata di caricamento durante l'esecuzione dei task asincroni.

Le chiamate alle procedure remote devono avvenire in background per non bloccare l'interfaccia utente e per evitare che, in caso di problemi, i tempi richiesti dalla connessione si allunghino e che quindi venga generato un errore di *Application Not Responding*.

Per questi motivi ogni chiamata alle Cloud Functions è effettuata in maniera asincrona.

La classe per interfacciarsi con le Cloud Functions è *FirebaseFunctionHelper*.
In particolare ogni metodo di questa classe gestisce l'interfacciamento con una singola Cloud Function e restituisce un oggetto di tipo *Task<T>*, dove *T* è il tipo di dato che restituisce la chiamata alla Cloud Function di interesse.

Implementazione dell'app – Xamarin

Nella seconda parte del progetto la stessa app è stata realizzata in un'ottica cross-platform. L'obiettivo è quindi quello di sviluppare un'unica app in grado di essere eseguita su sistemi operativi diversi (Android e iOS), minimizzando il codice specifico per le singole piattaforme.

In quest'ottica è stata però alla fine implementata soltanto la versione Android dell'app.

L'app cross-platform particolarizzata per Android è stata realizzata con il framework Xamarin e utilizzando per la condivisione del codice l'approccio PCLs (o meglio .NET standard).

È stata quindi realizzata una soluzione contenente due progetti:

- XamarinApp: progetto cross-platform contenente logica di business, accesso ai dati e UI cross-platform in Xamarin.Forms
- XamarinApp.Android: progetto specifico per la piattaforma Android con codice nativo

Componenti

Il progetto cross-platform si compone di una serie di file raggruppabili concettualmente nelle seguenti categorie:

- Models: insieme di classi che modellano i dati restituiti dalle interrogazioni del database esterno (Spese)
- Pages: insieme delle schermate dell'app; ogni schermata si compone di un file XAML che definisce layout ed elementi della pagina e un file C# che ne gestisce la logica;
- Classe App: è la base dell'app cross-platform, referenziata nel singoli progetti; permette di definire la pagina iniziale dell'app;
- View Model : utilizzati per il binding dei dati dalla vista al model e viceversa.

Gestione login

La gestione del login in Xamarin si basa sulle shared preferences, che ci permettono di memorizzare i dati dell'utente e accedervi da ogni punto dell'applicazione.

Quindi se la procedura di login va a buon fine, vengono memorizzati i dati dell'utente nelle preferences e viene assegnato un token (Token Bearer) che permette di autenticarsi alle Cloud Functions.

Con i dati memorizzati, ai prossimi avvii dell'app, saremo indirizzati immediatamente nella homepage, a meno che non si faccia il log out, il quale elimina i dati memorizzati nelle preferences e pulisce lo stack delle pagine.

Conclusioni

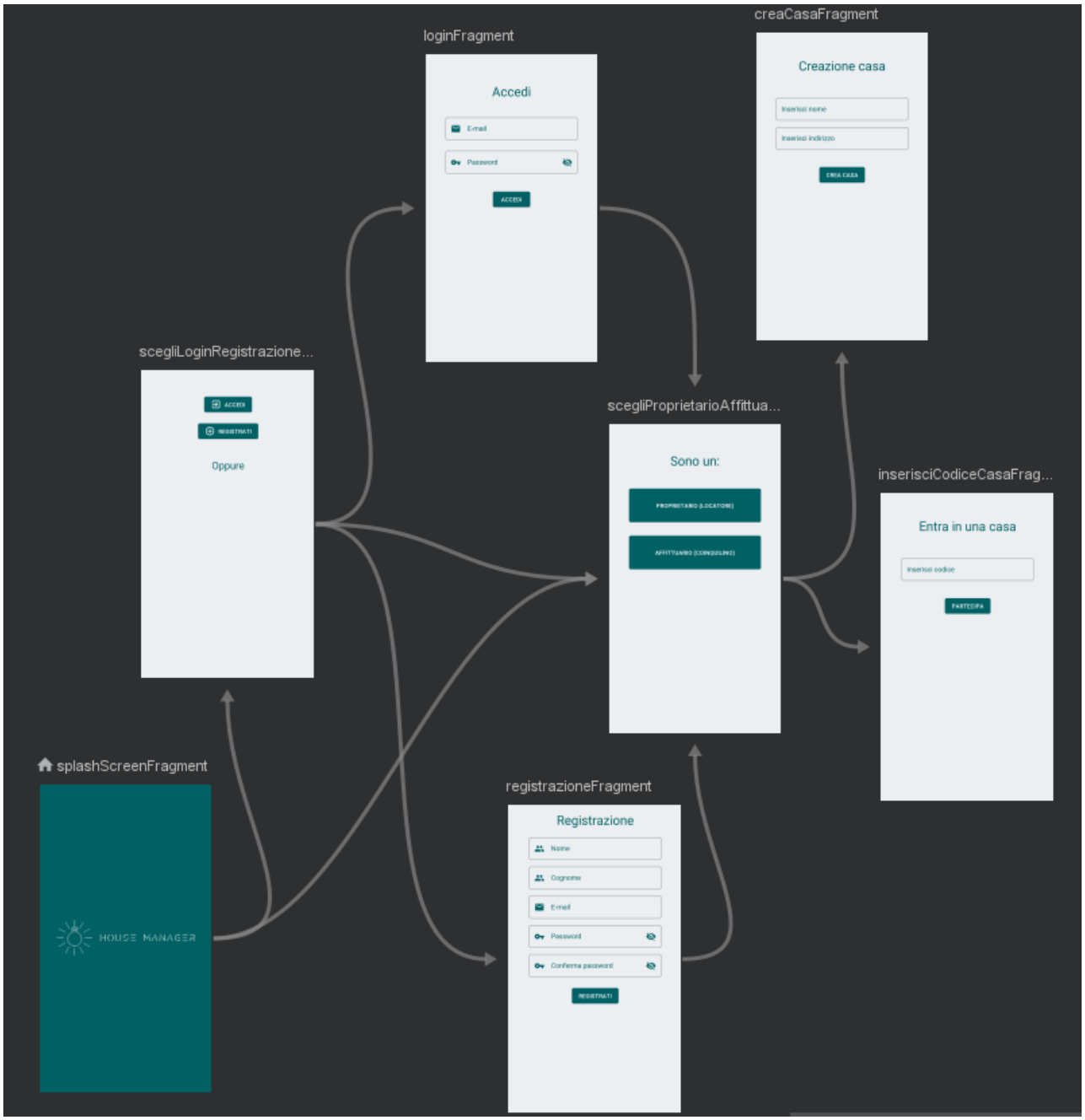
User Experience

Durante lo sviluppo il nostro scopo è stato quello di sviluppare un'applicazione che fosse quanto più vicino possibile ad un'applicazione reale, motivo per il quale abbiamo prestato particolare attenzione ad ottimizzare l'esperienza utente grazie all'utilizzo di alcune componenti grafiche.

Navigation

La Navigation Component permette, nella fase di registrazione/login di navigare, entrare e uscire tra le diverse pagine che compongono l'intera sezione.

Tale componente garantisce la possibilità di inserire animazioni di transizione tra i componenti che la definiscono, così che l'utente possa tener traccia dei passaggi svolti.



Time Picker e Date Picker

I Time Picker e Date Picker possono essere classificati come Widget per Interfacce Utente che consentono all'utilizzatore del servizio di selezionare, nel caso dei Date Picker, la data da un calendario e, nel caso dei Time Picker, l'ora da un orologio.

Oltre che essere componenti grafici, sono al tempo stesso funzionali, in quanto permettono di "validare" l'inserimento effettuato dall'utente prima che questo di fatto avvenga, non consentendogli di indicare date o orari con formati non congrui con il tipo di campo.