REPORT SU MATCHMATES

Progetto per il corso Laboratorio progettazione sistemi mobile e tablet, 2022 – 2023

Componenti del gruppo:

Angelo Bottazzo mat. 221415
Leonardo Bottona mat. 221997

UNIVERSITÀ DEGLI STUDI DI TRENTO

Report su **MatchMates**

Progetto per il corso Laboratorio progettazione sistemi mobile e tablet, 2022 – 2023

1. Introduzione

MatchMates è volta a facilitare l'organizzazione di partite in compagnia, aiutando a trovare persone, tra i tuoi amici o possibili nuove conoscenze, a cui interessa giocare allo stesso sport, e suggerendo un campo da gioco.

Nello specifico, l'app è un social che riunisce le persone intente a fare una partita, permettendo creare annunci con diversi livelli di visibilità: privati per restringere la partecipazione alle sole persone scelte, gli amici, oppure pubblici per dare la possibilità a chiunque di unirsi e fare nuove conoscenze. Inoltre, gli utenti potranno trovare campi da gioco esistenti o suggerirne di nuovi.

2. Identificazione del segmento utente

Il target principale sono utenti giovani e sportivi, o chiunque voglia passare un pomeriggio all'aperto. Tuttavia, gli sport inclusi sono molti, chiunque pratichi uno sport previsto dall'applicazione può trovarla utile.

Abbiamo riscontrato diversi problemi nel modo tradizionale di organizzare un semplice pomeriggio di gioco con amici: primo tra tutti, trovare il numero sufficiente di giocatori; in secondo piano, organizzarsi nelle chat di gruppo risulta dispersivo: si perdono di vista le informazioni essenziali per mettersi d'accordo, come data e luogo, o chi parteciperà; come ultimo problema, ma spesso ostacolo principale, la difficoltà a trovare un posto libero e adatto a svolgere l'attività accordata.

3. Stato dell'arte

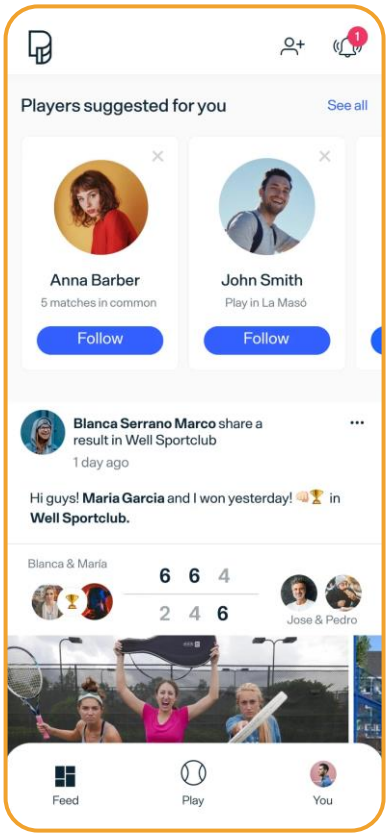
Ci sono diverse applicazioni nel PlayStore e nell'AppStore che al primo sguardo sembrano simili a MatchMates. Approfondendo, si nota che sono sviluppate con un'idea di base diversa.

Playtomic

Permette di prenotare un campo presso centri sportivi che si sono iscritti al servizio, non è possibile quindi organizzarsi per una partita nei campi pubblici (ad esempio quello di un'area verde di una città), o quelli non ancora presenti sulle mappe.

Gli sport sono limitati a tennis, padel e calcio.

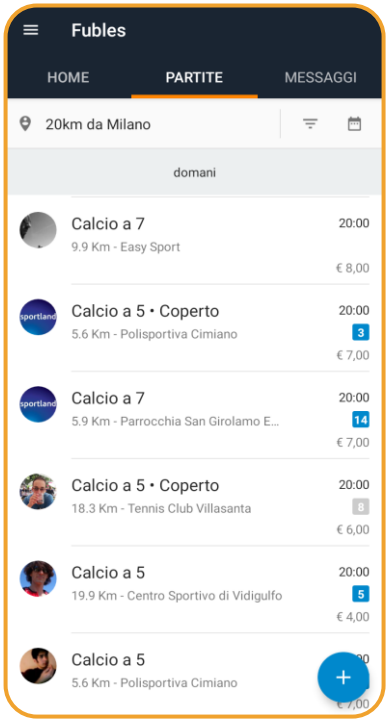
È un servizio orientato maggiormente alla prenotazione di strutture che all'organizzazione di partite "da zero".



Fubles

Fubles ha molte funzioni in comune con l'app in progettazione, compreso l'inserimento di strutture pubbliche, ma è specializzata per il calcetto.

Nell'organizzazione della partita, in Fubles vengono chiesti dettagli molto specifici, ad esempio le squadre, o le posizioni di ogni giocatore. MatchMates non aderisce con tanta precisione alle diverse esigenze dei vari sport, ma lascia agli utenti definire tali dettagli nel modo che ritengono più opportuno.

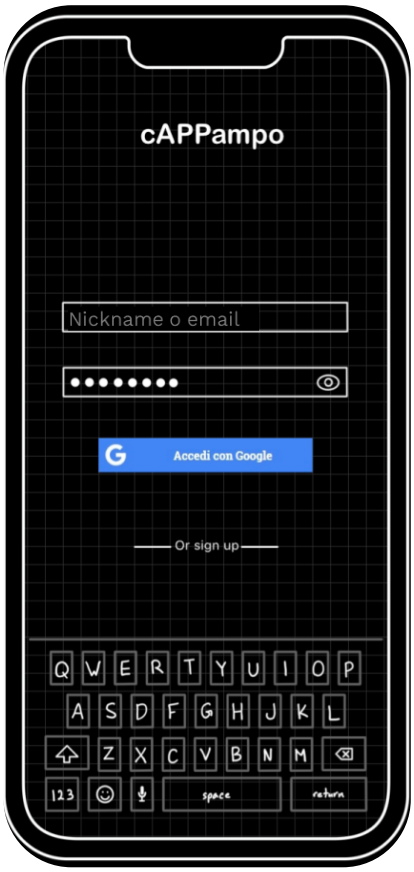


4. Wireframe e navigazione

Login

Accesso tramite: nickname o e-mail e password, anche con Google.

Per la registrazione: nome, cognome, data di nascita, zona preferita, (Sport preferiti)

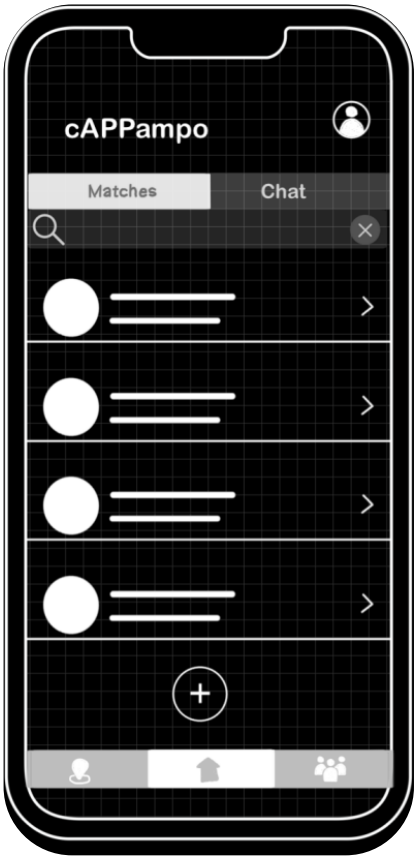


Home > Partite

Elenco di partite: in testa ci sono le partite a cui si è stati invitati, quelle in cui si partecipa, e sotto quelli suggeriti in base al luogo preferito.

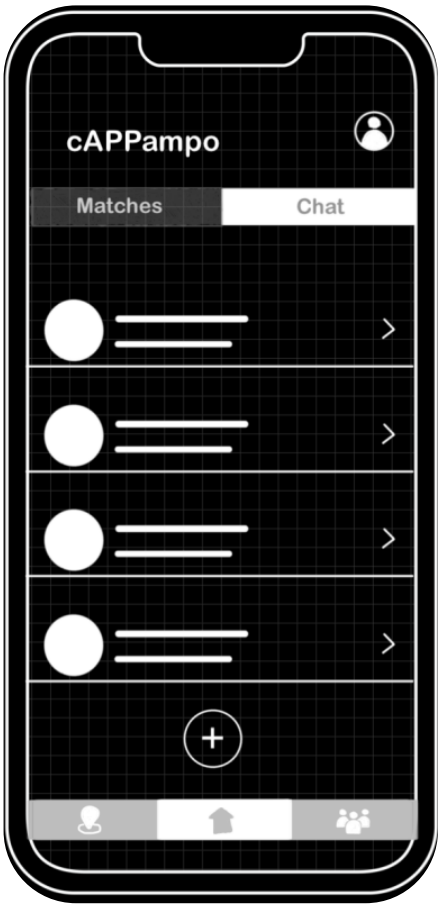
La barra di ricerca apre i filtri per sport e luogo.

Il pulsante + in basso porta alla creazione di una partita.



Home > Chat

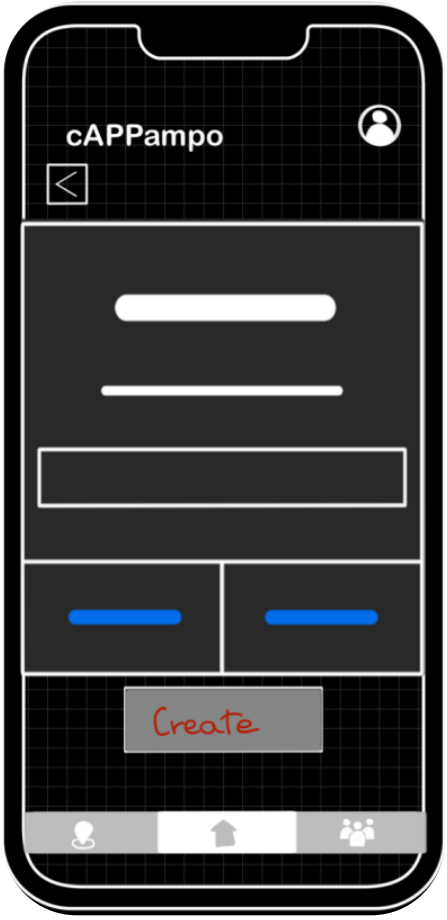
Da qui si possono vedere velocemente tutte le chat relative alle partite in cui si partecipa o con altri utenti.



Creazione partita

Inserimento di dettagli:

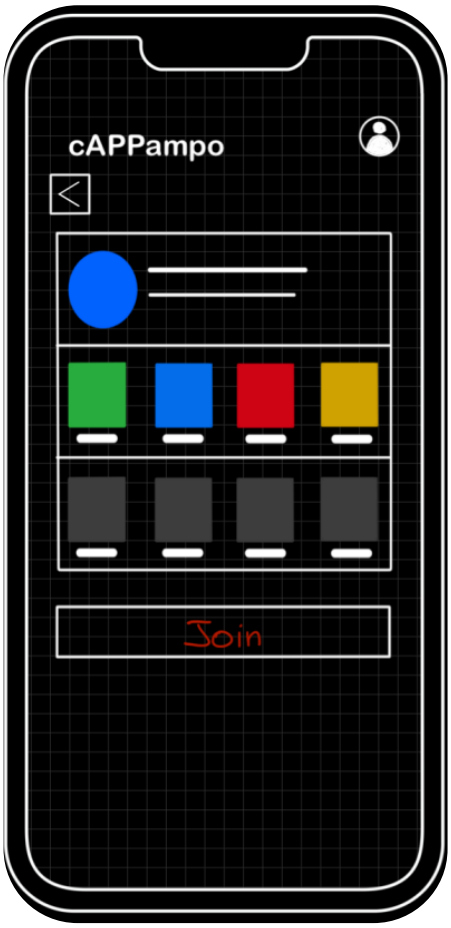
- Nome partita
- Data e ora
- Sport
- Numero partecipanti
- Partecipanti da invitare
- Visibilità: solo invitati / amici / pubblico
- Descrizione
- Zona o luogo



Partecipazione a partita

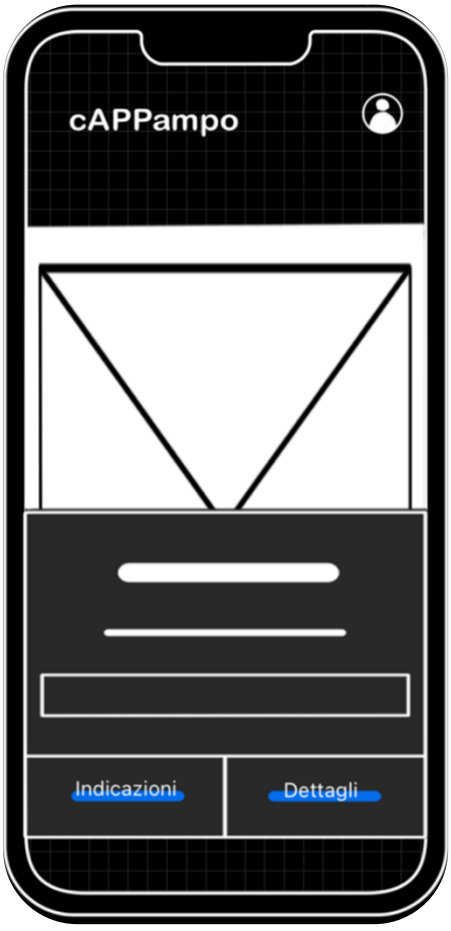
Anteprima dettagli:

- Nome partita
- Data e ora
- Zona o luogo
- Sport
- Partecipanti
- Posti rimanenti
- Descrizione



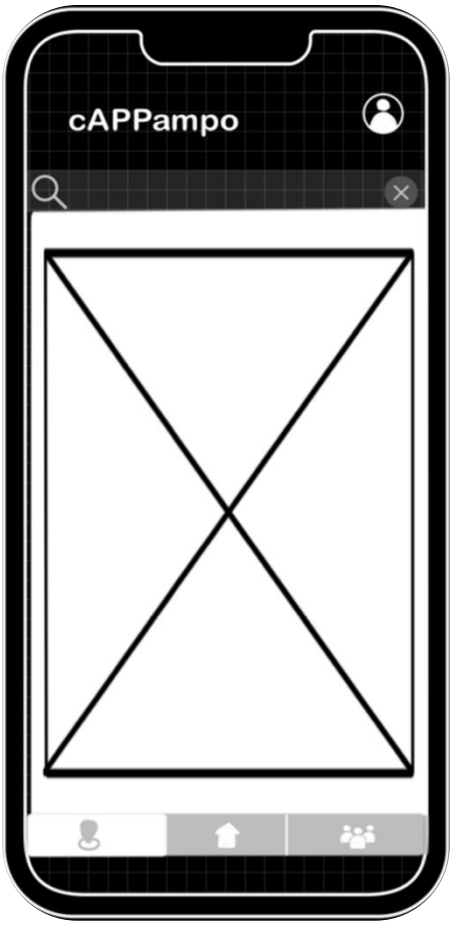
Partecipazione a partita > Anteprima luogo

- Mappa
- Nome luogo
- Distanza dal luogo
- Pulsante per indicazioni verso il luogo
- Pulsante per dettagli sulla struttura



Home > Mappa

- Ricerca di luoghi filtrati per sport
- I luoghi in cui praticare sport sono inseriti dalla community.
- Dai dettagli di un luogo, si può creare una partita.



Home > Amici

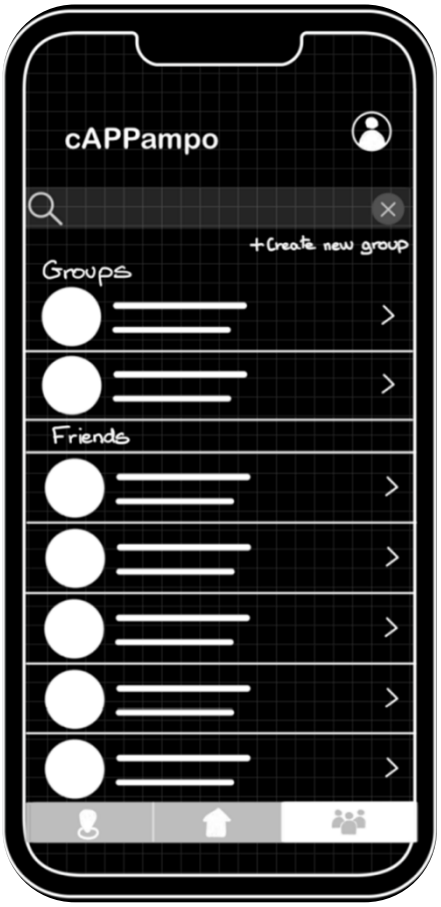
In testa:

- Gruppi, che permettono di invitare velocemente diverse persone

Poi:

- Amici

Un utente può vedere le partite degli amici (se la visibilità della partita lo permette).

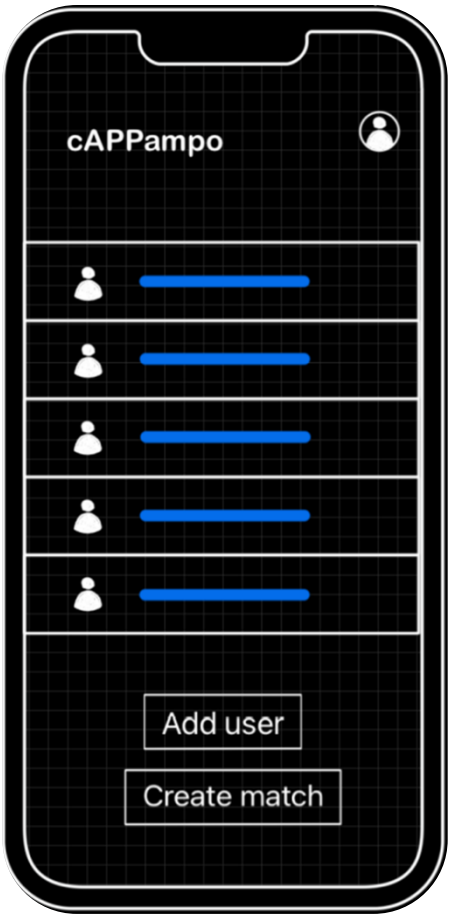


Amici > Dettagli gruppo

- Elenco dei partecipanti
- Pulsante per aggiungere persone al gruppo

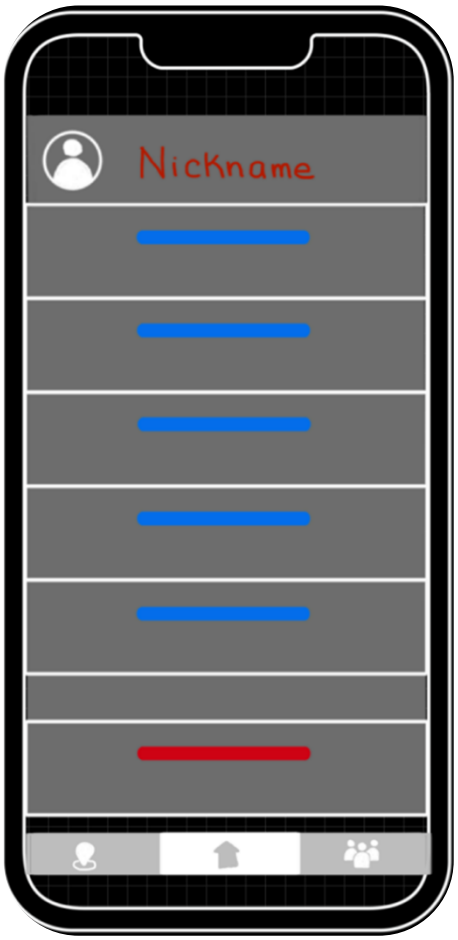
Se una persona non è già amica, serve inserire il nickname esatto per trovarla e aggiungerla

- Pulsante per creare velocemente una partita con loro



Impostazioni profilo

- Foto profilo
- Nickname
- Tema sfondo
- Gestione dei dati (zona preferita, data di nascita, ecc.)
- Cancellazione account



5. Architettura

L'app è costruita sul framework Flutter, e si appoggia alla piattaforma Supabase per le funzionalità legate al cloud. Pressoché tutti i dati dell'app risiedono su Supabase, che fa uso del DBMS PostgreSQL.

Autenticazione

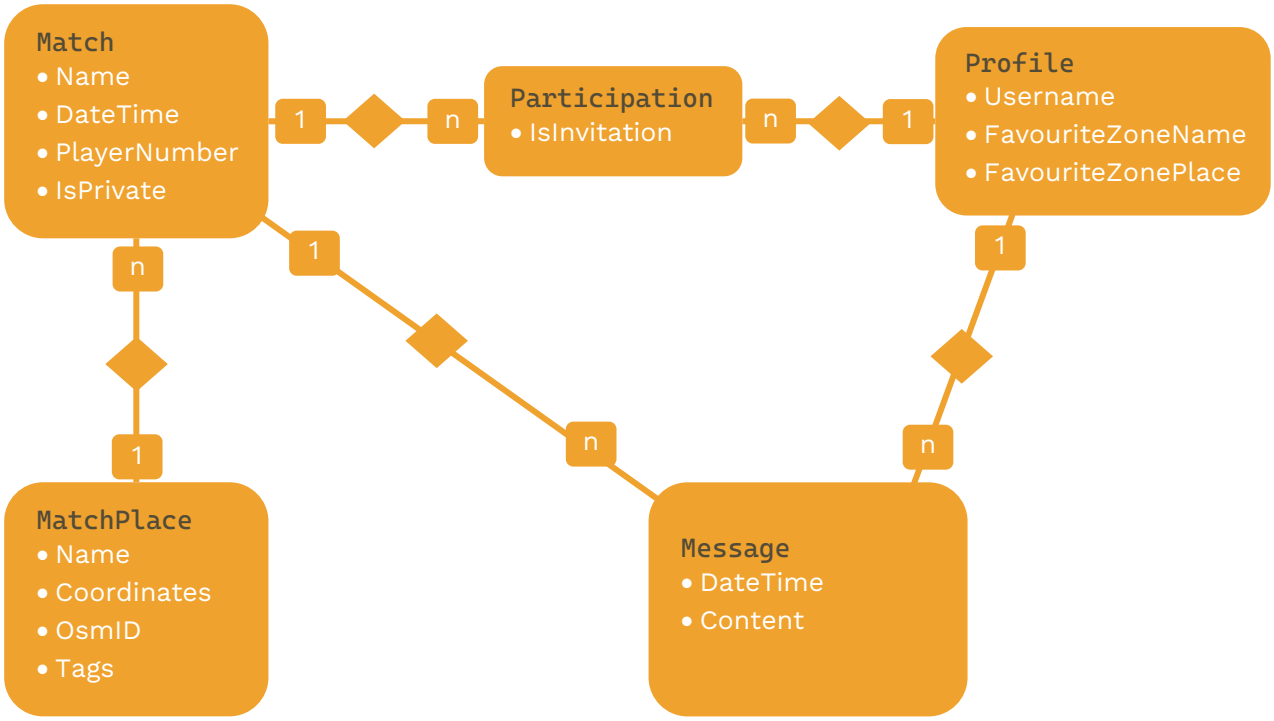
Per l'autenticazione, a differenza del wireframe, si è optato per il solo login tramite Google. Supabase e la sua libreria client per Flutter interagiscono con il provider (in questo caso Google) per creare l'account in Supabase ed eseguirne l'autenticazione nel dispositivo dell'utente.

Dopo aver richiesto di fare il login in MatchMates, viene aperta la pagina web di autenticazione del provider. Supabase ottiene i dati sull'account e reindirizza il browser ad un URL che apre MatchMates e che contiene anche un token di autenticazione. Questo tipo di link, detto *deep-link*, a differenza dei classici link `https://`, ha questa forma:
`io.supabase.matchmatesflutter://login-callback/#access-token=1234abcd…`

Il protocollo `io.supabase.matchmatesflutter`, nel dispositivo utente, è associato a MatchMates (grazie a un'informazione inserita nel manifesto dell'app), così quando il browser proverà ad aprirlo, il dispositivo aprirà MatchMates.

Database

In seguito, sono riportati i modelli e relazioni più interessanti all'interno del database.



Match, Participation e Profile
Una delle funzioni più importanti dell'app è raccogliere le partecipazioni ad un match. Questo è fatto grazie a `Participation`, che mette in relazione un utente con un match (all'atto pratico implementa una relazione n:m).

Più interessante è ciò che accade quanto un utente invita a un match un altro utente. Un invito è salvato come una normale partecipazione, anche perché così all'invitato viene riservato un posto senza dover programmare ulteriori controlli. Tuttavia, l'attributo `IsInvitation` è impostato a `true`, ciò dà modo all'app di mostrare l'interfaccia atta a rifiutare o confermare l'invito. In caso di rifiuto, la riga viene rimossa; in caso di accettazione, `IsInvitation` viene impostato a `false` così si trasformerà in una normale partecipazione.

La chat del match
Ogni match ha una la chat, in modo che i partecipanti possano accordarsi sui dettagli che l'app non chiede di impostare. Per mostrare i nuovi messaggi in tempo reale, è stata usata la funzionalità Realtime di Supabase, che notifica cambiamenti nelle tabelle del DB ai client connessi.

I MatchPlace
Ogni match è associato ad un MatchPlace, che rappresenta un luogo sulla mappa, con un nome ed eventualmente dei dettagli (ad esempio, su illuminazione, superficie del campo...).

Notifiche
Per inviare le notifiche agli utenti interessati su avvenimenti come accettazione o rifiuto di inviti, messaggi ricevuti, oppure modifiche o eliminazioni di match, sono stati usati diversi strumenti in combinazione.

Firebase Cloud Messaging (FCM)
È un servizio di Firebase che si occupa di recapitare un payload di notifica ad un dispositivo. Il payload di notifica viene inviato a tale servizio attraverso una richiesta HTTP POST, e il dispositivo è identificato da un token generato durante il login nell'app (salvato in un'apposita tabella, associato al relativo utente).

Payload di notifica
È un oggetto `JSON` che rappresenta i dati necessari a creare la notifica sul dispositivo. Contiene:
- Il token del destinatario
- Il titolo e il corpo della notifica (possono essere una stringa o l'ID della stringa localizzata)
- L'URI da aprire nel dispositivo al tocco della notifica

Trigger
PostgreSQL prevede la possibilità di impostare l'esecuzione di una funzione ogni volta che accade un evento specifico nel database, attraverso i `trigger`.

Tabella delle notifiche
Nel database è presente una tabella che contiene le notifiche ricevute da un utente. Conservare le notifiche è necessario per poterle mostrare anche in un elenco all'interno dell'app.

Un esempio
Viene accettato un invito: ciò lancia l'esecuzione del trigger relativo agli `UPDATE` nella tabella `Participations`, che genera un payload di notifica e lo aggiunge tabella delle notifiche. Un ulteriore trigger legato all'`INSERT` nella tabella delle notifiche invia, tramite una richiesta POST, il payload appena aggiunto all'endpoint di FCM, che si preoccuperà di recapitarlo al dispositivo interessato.

6. Implementazione

La fase più turbolenta dello svolgimento del progetto è stata l'implementazione. La ragione, oltre alle consuete difficoltà legate alle specifiche del funzionamento degli strumenti a disposizione, si può trovare in alcune limitazioni inizialmente poco evidenti nelle tecnologie scelte, che ci hanno costretto a dei cambi di rotta. In seguito, sono descritti i ragionamenti che hanno portato alle scelte effettuate, e ai successivi ripensamenti.

AndroidX e Firebase

Le lezioni vertevano attorno allo sviluppo nel framework nativo di Android, usando l'IDE Android Studio. La scelta naturale è stata di seguire tali orme, scegliendo il linguaggio Java, anche per l'esperienza pregressa grazie ad un corso svolto nel semestre precedente.

Un primo scoglio, per uno dei due componenti del gruppo, è stata l'assenza di un dispositivo fisico su cui eseguire l'app. Fortunatamente, Android Studio offre un simulatore.

Per quanto riguarda l'interazione con il cloud, è stato scelto Firebase. Firebase è conosciuto per essere semplice da integrare nelle applicazioni Android, e in effetti così è stato.

Abbiamo apprezzato questi aspetti:

- La configurazione del progetto in Android Studio è guidata
- La serializzazione e la deserializzazione degli oggetti è quasi automatica, grazie alla reflection di Java
- Si possono facilmente configurare eventi che aggiornino i dati locali quando avvengono cambiamenti nel cloud
- Non è necessario preparare uno schema del database, poiché si basa su JSON
- Le policy di sicurezza rendono sicuro il database nonostante l'assenza di un vero e proprio server API
- L'autenticazione tramite Google si integra facilmente.

Realizzato il salvataggio e visualizzazione dei match, lo sviluppo è proseguito sulle funzionalità interazione tra gli utenti. Ciò ha richiesto di rivedere le fondamentali relazioni tra i modelli dei dati:

- un utente può creare vari match
- ai match possono partecipare vari utenti
- un match è di un solo sport
- un match si svolge in un luogo
- In un luogo si possono praticare diversi sport
- In un luogo possono verificarsi vari match
- Ogni utente ha diversi amici
- Un amico può essere invitato solo una volta ad un match

A questo punto, un aspetto di Firebase si è rivelato svantaggioso: non esiste il concetto di "relazione" tra i dati, e l'elenco qui sopra assomiglia pericolosamente alle indicazioni per costruire un database relazionale. Tuttavia, un documento (un'entità identificata da un ID si chiama così in Firebase) può riferirsi ad un altro tramite il suo identificativo, o mantenendone una copia.

Purtroppo, salvare l'identificativo significa che è necessario fare più query per ottenere un'entità nel suo complesso[1], e duplicare un'entità richiede che ogni cambiamento debba essere applicato su ogni sua istanza.

Si è fatta strada la sensazione che optare per un database non relazionale non fosse stata la scelta migliore.

Nel valutare la transizione ad un database SQL, la volontà era di mantenere un'architettura simile a quella di Firebase, ovvero priva di un server intermedio che gestisca le richieste dei client e che esegua le query nel database: fare invece così avrebbe richiesto di trovare un servizio di hosting (ulteriore a quello del database) e creare per ogni tipo di query di cui il client avrebbe avuto bisogno una nuova funzione API. Dopo varie ricerche, è stato deciso di usare Supabase.

Supabase

Si tratta di un servizio nato dall'intenzione di creare un'alternativa a Firebase indipendente da Google. A differenza di Firebase, Supabase sfrutta il database relazionale PostgreSQL, e traspone varie funzioni della controparte di Google: la gestione dell'autenticazione tramite OAuth2.0, la segnalazione di eventi in tempo reale, regole per autorizzare l'accesso ai dati (RLS rules) e un SDK per chiamare facilmente le funzioni dai client. Purtroppo, l'SDK non era disponibile per Java, ma solo per JavaScript e Flutter.

Gli ostacoli di AndroidX

Fino a quel momento, l'esperienza durante lo sviluppo non era stata molto gratificante. Oltre alla seccatura di dover usare per forza un emulatore e non poter utilizzare l'app che si stava creando per un componente del gruppo, il framework nativo di Android è parso limitante sotto vari aspetti.

Il passaggio di dati da una Activity a un'altra

MatchMates è composta di diverse schermate, molte delle quali devono scambiarsi oggetti, ad esempio l'activity per scegliere un luogo su una mappa, per visionare i dettagli di un match o le statistiche di un utente. Tutte le classi che vengono scambiate devono implementare l'interfaccia `Parcelable`, che permette alla piattaforma di serializzare e deserializzare i dati più velocemente di quanto riesca a fare la controparte nativa di Java. Android studio riesce a scrivere il boilerplate automaticamente, a patto che tutti i campi della classe implementino `Parcelable`, cosa che spesso non si verificava, anche con tipi di dati nativi.

Le liste

Molte viste richiedevano un elenco, ad esempio di match, di utenti o di luoghi. Il framework mette a disposizione la `RecyclerView`, ma il suo utilizzo è complesso e spesso porta a errori difficili da scovare.

Le operazioni asincrone

MatchMates scambia spesso dati con il server: tali richieste devono essere fatte in modo asincrono, per non bloccare il thread principale e causare ritardi nell'interazione. Il modo consueto di rispondere ad una chiamata asincrona in Java è tramite dei listener, ma questi rendono il codice meno ordinato, e meno chiaro il flusso di esecuzione del codice.

[1] Questa pratica non è sempre attuabile, ecco un esempio: se si volessero recuperare i partecipanti ad un match a partire dalla lista dei loro ID, verrebbe naturale scaricare gli utenti il cui ID è contenuto in un array tramite la clausola `.WhereIn("id", ID_array)`, ma per una limitazione vengono considerati solo i primi 10 elementi di `ID_array`

Flutter

Tutti questi aspetti ci hanno spinto a fare di necessità virtù. Anche se per usare l'SDK di Supabase sarebbe stato necessario trasporre il lavoro già fatto in un progetto basato su Flutter, l'alleggerimento che questo framework ci avrebbe portato è stato giudicato vantaggioso, nonostante l'onere del porting. In più, Flutter è multipiattaforma.

L'architettura

Flutter supporta il linguaggio Dart, e si basa su una gerarchia di `Widget` per costruire l'interfaccia. Un `Widget` si può immaginare come una ricetta che verrà usata durante l'esecuzione per disegnare i vari elementi a schermo. Per definire un `Widget`, è sufficiente scrivere una classe in Dart che erediti da `StatefulWidget` o `StatelessWidget`. Tali classi astratte chiedono di implementare il metodo `build()`, che ritorna il `Widget` che si vuole costruire.

Uno `StatefulWidget`, rispetto a uno `StatelessWidget`, prevede la possibilità di mantenere dei campi che ne rappresentino lo stato. Per fare un esempio, uno `StatefulWidget` può essere un widget che mostra i match vicini, il cui stato è rappresentato da due campi: un booleano che indica se è in corso il download chiamato `isDownloading`, e una lista che mantiene i match scaricati. Facendo un override del metodo `initState()`, si può avviare la procedura che scarica i match dal database quando il `Widget` viene costruito per essere presentato. Questa procedura funzionerebbe così:

1. Impostare `isDownloading` a `true`
2. Avviare il download
3. Aspettare la fine del download grazie ad `async` / `await`
4. Riempire la lista di match con i dati scaricati
5. Impostare `isDownloading` a `false`

Ogni volta che lo stato cambia, la vista viene ricostruita automaticamente in modo che si riflettano i cambiamenti dello stato. Si può ad esempio mostrare un'animazione di caricamento solo mentre `isDownloading` è `true`, inoltre i match mostrati si aggiornano automaticamente quando cambia la relativa lista nello stato.

È interessante notare la presenza di `async` e `await`: in particolare anteponendo `await` alle chiamate asincrone, il flusso di esecuzione si interrompe finché la chiamata asincrona non viene risolta. Questo semplifica di molto la scrittura di codice asincrono, perché non richiede un cambio di paradigma rispetto alla normale programmazione, cosa che invece Java fa con i listener. `async` è la keyword da anteporre ad un blocco di codice per segnalare che è asincrono.

Infine, Flutter astrae alcune delle complicazioni che, sviluppando in AndroidX le funzioni di cui sopra, si incontrano. Ad esempio, non servono particolari attenzioni nello spostare dati tra una pagina e l'altra, ed esistono vari widget, sia nativi sia preconfezionati dalla community, per implementare funzioni comuni come elenchi, ricerca o filtraggio. La documentazione è chiara e ricca di esempi. Purtroppo, risulta facile intrecciare codice per generare l'interfaccia e per eseguire operazioni, poiché scritti nello stesso linguaggio e potenzialmente presenti nello stesso file. Da una parte ciò semplifica la modifica dell'interfaccia da parte del codice, dall'altra ostacola l'applicazione del concetto della *separation of concerns*, con classi che senza adeguate accortezze possono diventare molto grandi e difficili da consultare.

Google Maps e OpenStreetMaps

Durante lo sviluppo dell'app Android, la scelta della libreria per le mappe è ricaduta su Google Maps, per la facilità dell'integrazione e per la grafica snella. Si può facilmente interagire con i marker: questo fatto è stato d'aiuto per implementare la scelta del luogo in cui svolgere i match.

La controparte in Flutter di Google Maps è priva di questa funzione. Questo ha spinto a cercare delle alternative, e OpenStreetMaps ha attirato la nostra attenzione. Nonostante non sia stato possibile trovare una libreria che permettesse di premere sui luoghi per interagirci, è stato individuato un punto di forza nella quantità di dati inseriti dalla community.

Tramite Overpass API, è possibile ricercare i luoghi in base ai loro tag. I punti d'interesse legati allo sport risultano essere veramente tanti, è segnalata ad esempio anche la stazione di fitness nel lato nord del perimetro dell'edificio di Povo 1.

Il fatto che il widget per la mappa di per sé non sia interattivo (a meno della navigazione), ci ha costretto a scaricare preventivamente i luoghi legati allo sport nella sezione di mappa visibile, in modo da poter costruire dei widget, che l'utente possa azionare, sovrapposti alla mappa. Fare questo si è rivelato un vantaggio rispetto a ciò che ci era possibile su Google Maps: in quest'ultimo tutti i luoghi erano ugualmente visibili, con OSM invece è stato possibile evidenziare solo i luoghi in cui si può svolgere attività sportiva.

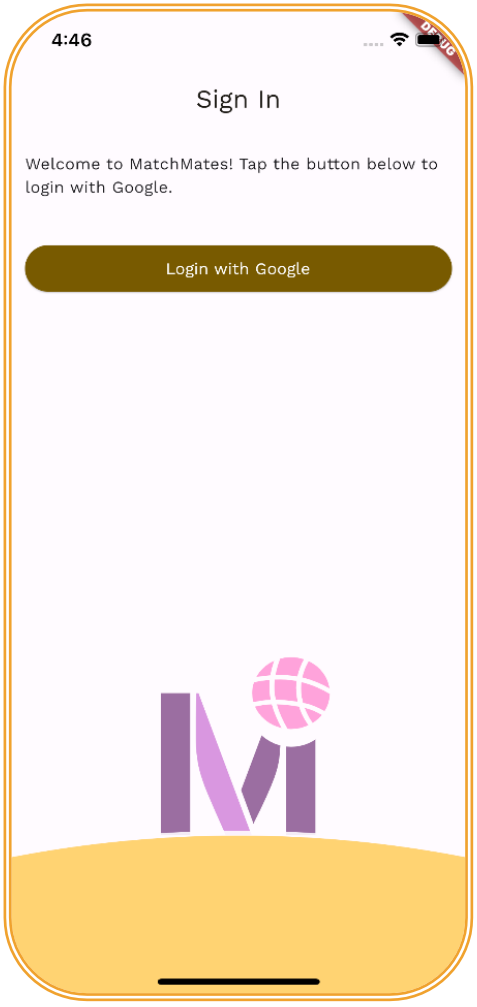
Il risultato

In seguito, ecco alcuni screenshot delle schermate salienti.

Login

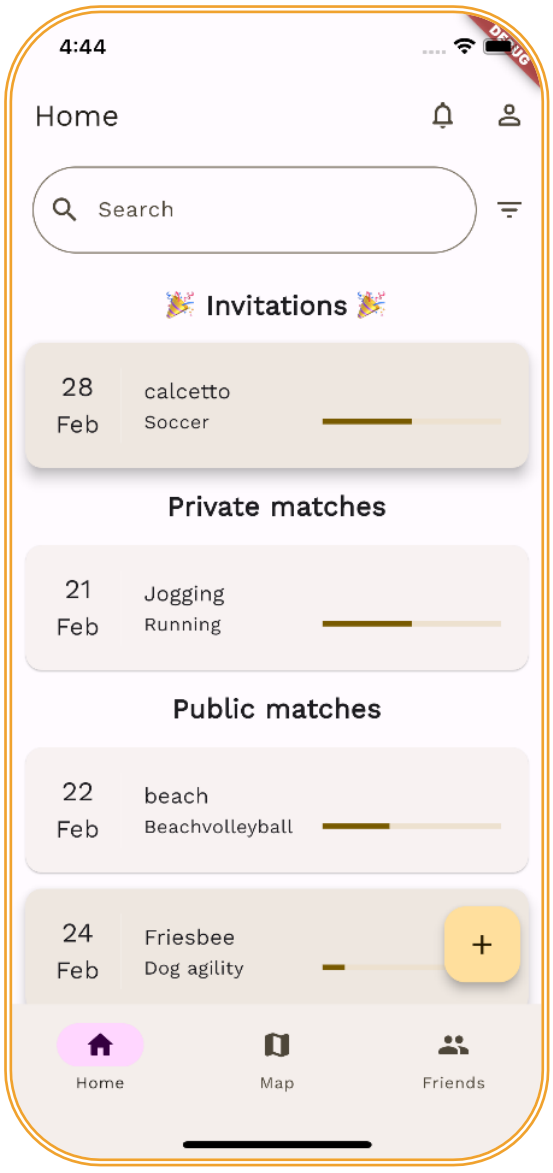
A differenza del wireframe, la schermata di login presenta solamente l'opzione per il login con Google.

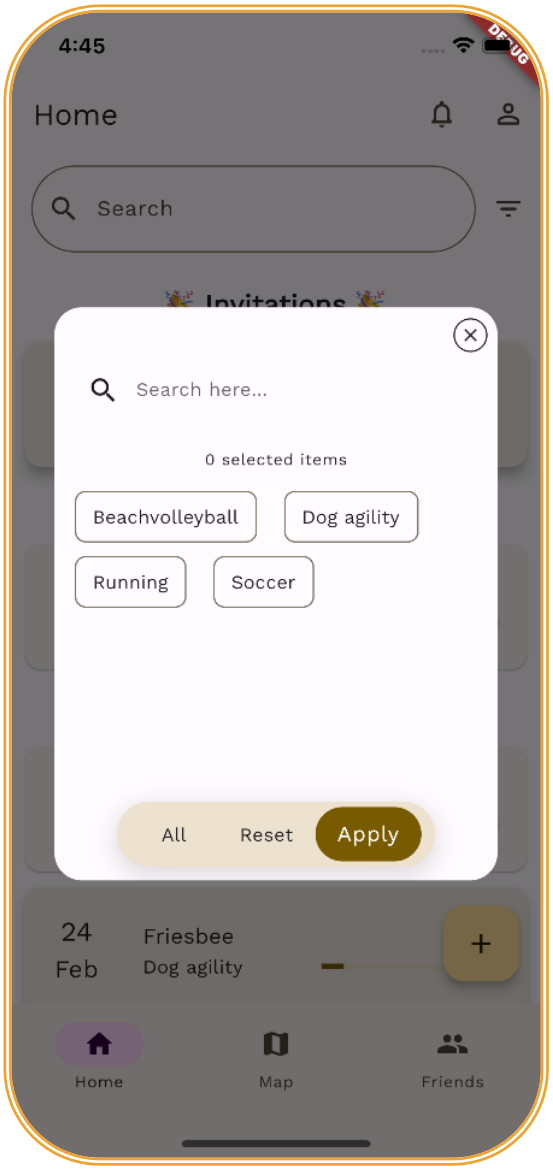
Il logo, in cui la palla rimbalza su due gambe della 'M', è stato animato su AfterEffects ed esportato nel formato vettoriale Lottie grazie all'estensione Bodymovin.



La schermata principale

Ci sono alcune differenze rispetto al wireframe. È stato aggiunto un pulsante per aprire l'elenco delle notifiche, che fanno notare l'entrata e uscita degli utenti dai match creati, ed è stato rimosso il pulsante delle chat. Anziché mostrarle tutte in un elenco, si è ritenuto più comodo collocarle nelle schermate dei relativi match.

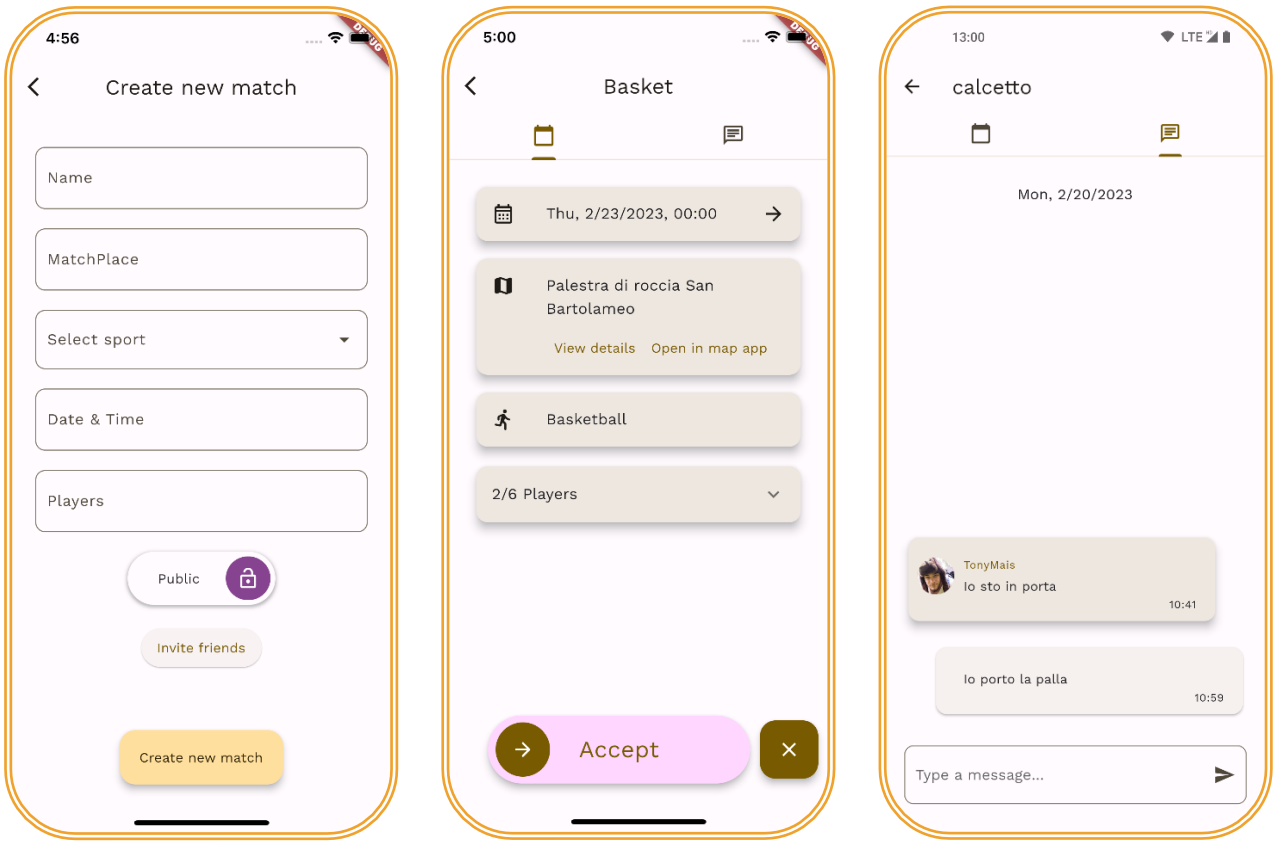




Creazione e dettagli del match

Rispetto al wireframe, è stato usato più spazio verticale.

L'accettazione oppure il rifiuto degli inviti avvengono rispettivamente tramite uno swipe sull'apposito controllo o una pressione su un pulsante.



La mappa

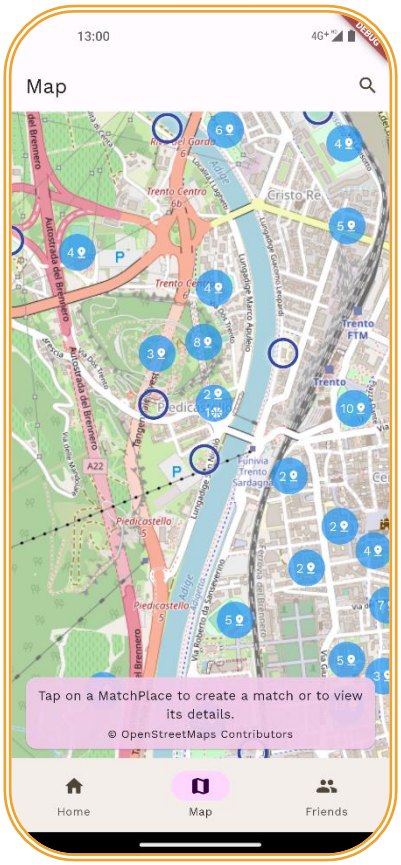
La mappa mostrata nell'app si appoggia a OpenStreetMaps, e ne evidenzia i punti di interesse in cui si può praticare sport (quelli con il tag `sport` impostato). I dettagli e luoghi inseriti dagli utenti vengono integrati ai dati scaricati da OSM.

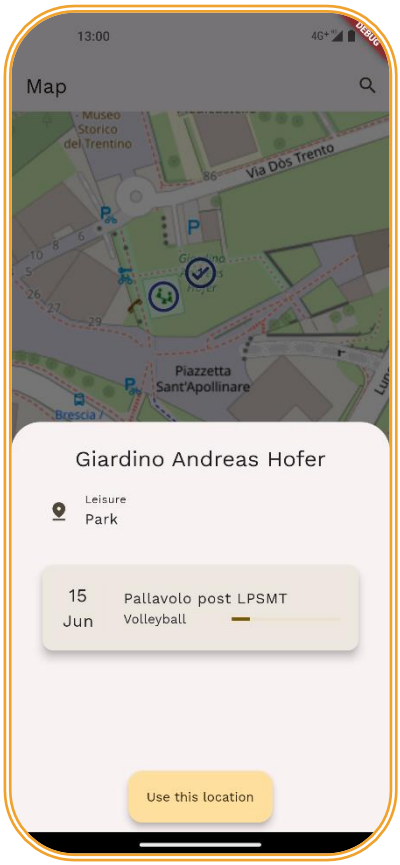
Se l'utente sceglie uno dei luoghi evidenziati, e il luogo non è già nel database, i dati di OSM sul luogo vengono copiati nel database per creare un MatchPlace. Se il luogo non ha un nome, viene chiesto di inserirne uno, per facilitare gli altri utenti a identificare il posto.

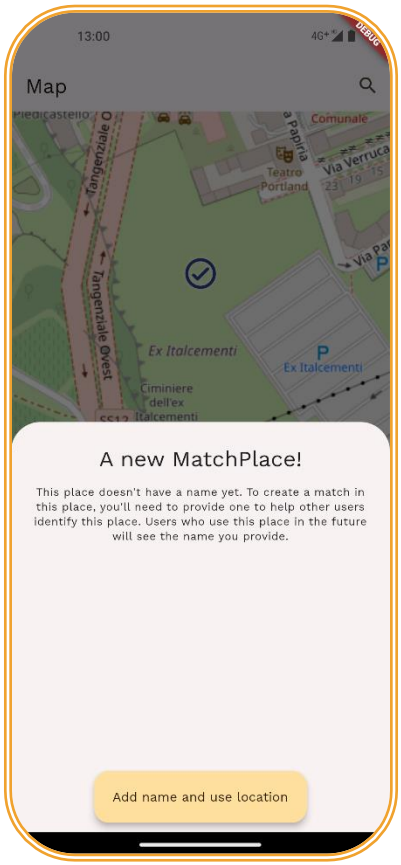
È anche possibile che un utente prema su un punto della mappa non evidenziato: potrebbe essere un luogo non adibito principalmente allo sport (come un bar che contiene un biliardo), o un punto completamente privo di informazioni (come il giardino dietro casa). Per verificare se il posto è un punto di interesse, le coordinate in cui l'utente ha premuto vengono inviate a Nominatim per un reverse lookup. Se si trovano dei dettagli, questi vengono copiati nel database per creare un MatchPlace, altrimenti, si chiede un nome all'utente e viene comunque salvato come MatchPlace.

Se un luogo inserito manualmente dagli utenti contiene dei match pubblici, allora il luogo viene evidenziato al pari degli altri sulla mappa.

I punti di interesse molto vicini sono raggruppati in un cerchio, che indica quanti posti vi sono stati aggregati e il totale di match in quei luoghi.

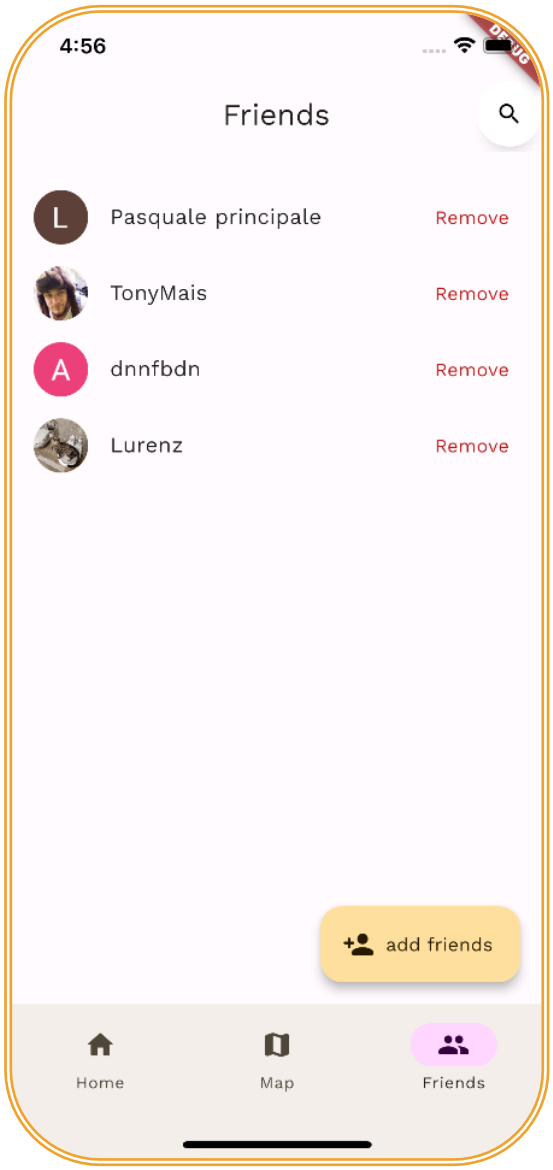


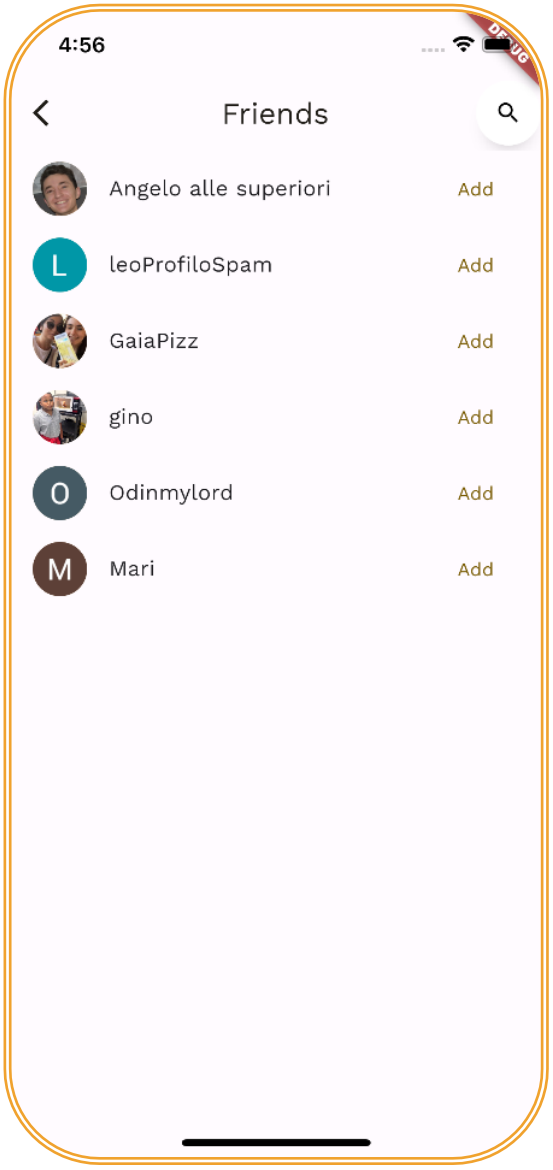




Le schermate degli amici

A dispetto del wireframe, le funzionalità dei gruppi sono state ritenute di priorità più bassa e sono state tagliate dalla release.

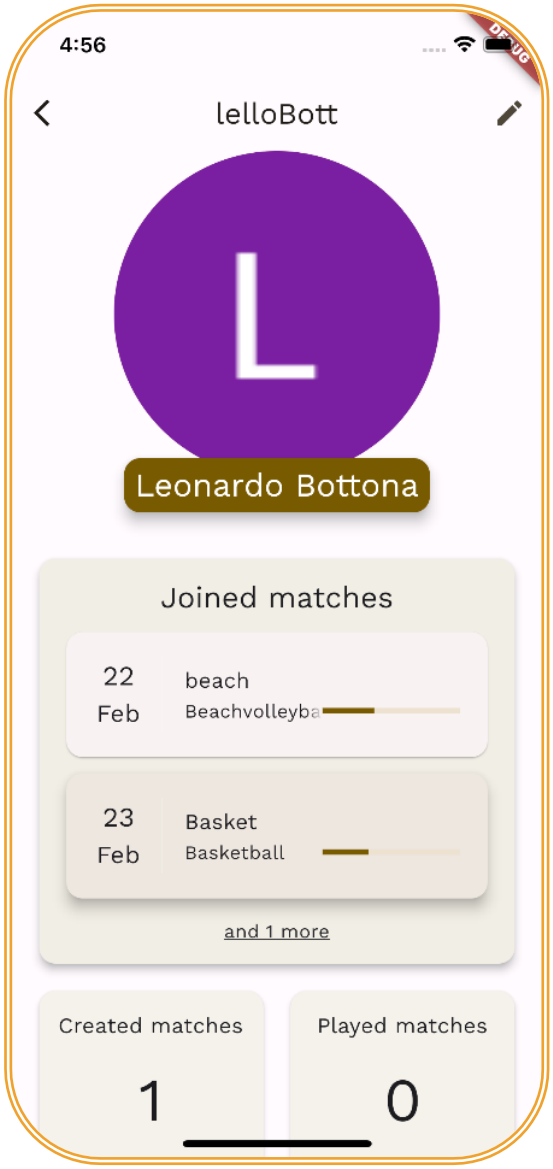


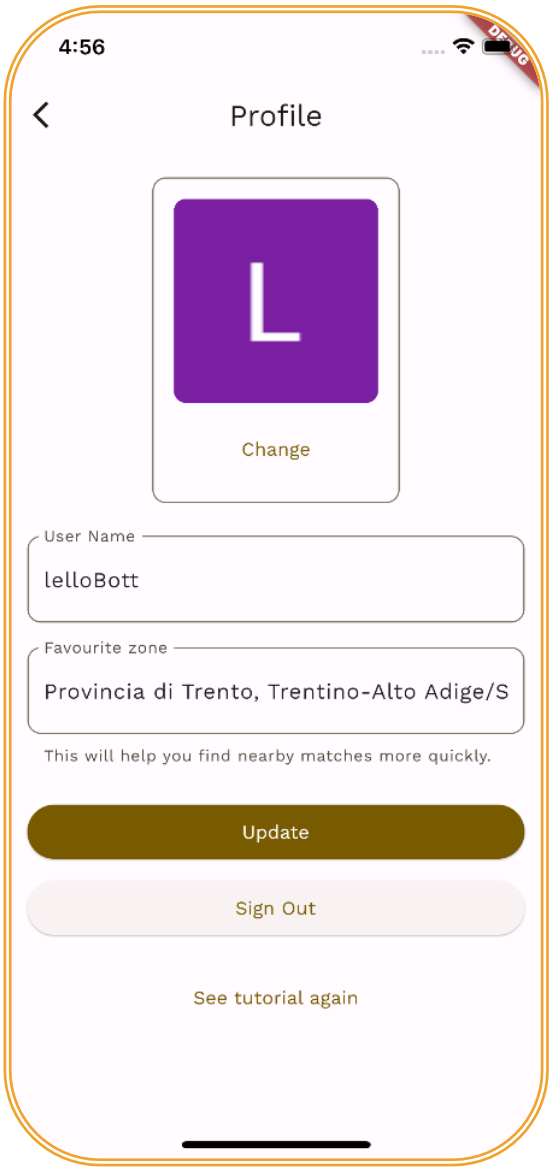


Statistiche e impostazioni profilo

È stata aggiunta una schermata sulle statistiche degli altri utenti.

Nella schermata delle impostazioni del profilo, sono stati ridisposti gli elementi.





7. Cambiamenti rispetto le versioni precedenti

Rispetto alla versione 1.0, la versione 2.0 ha visto queste modifiche:

- I dettagli su un match possono essere modificati dopo la creazione
- È possibile inserire liberamente lo sport del match
- I match si possono condividere attraverso un link
- Si ricevono notifiche per la rimozione o la modifica dei match
- Si ricevono notifiche, con la possibilità di rispondere, per i messaggi
- Nella mappa, è stato aumentato lo zoom minimo per il caricamento dei match
- Nella mappa, i MatchPlace vengono raggruppati se molto vicini

8. Valutazione

L'app è stata valutata fornendo l'APK ad alcuni compagni di corso e amici extra-universitari.

È stata apprezzata la cura della grafica e l'assenza di un menù hamburger; tuttavia, il tutorial iniziale è un po' troppo lungo e alcune cose che specifica sono già ovvie. Alcune volte le snackbar che confermano le operazioni ostacolano l'utilizzo dei pulsanti.

Per quanto riguarda la creazione dei match, non è prevista la specificazione di una durata stimata.

Sarebbe poi utile poter lasciare recensioni sul luogo usato.

La mappa è risultata un po' confusionaria, e lo scopo degli amici poco chiaro.

9. Analisi critica dei limiti dell'applicazione

Ci sono diversi aspetti che andrebbero rivisti prima di un'eventuale pubblicazione.

Come detto dai tester, al primo impatto non è chiaro a cosa servano gli amici e cosa comporti la loro aggiunta.

Per essere completamente rispettosi delle politiche di OpenStreetMaps, sarebbe necessario restituire alla community i dettagli aggiunti dagli utenti sulla mappa.

Esistono poi alcuni problemi di vulnerabilità nella base di dati, poiché nel codice dell'app è presente la chiave pubblica delle API. Questi problemi sono risolvibili impostando delle regole a livello di riga, come consigliato dalla documentazione di Supabase, ma ciò non è ancora stato fatto.

Sarebbe utile ottimizzare alcuni comportamenti dell'app nel download dei dati, alcune volte troppo frequenti.

Inoltre, l'interazione con le notifiche non è sempre affidabile.

Nonostante questi punti deboli, ci sentiamo di concludere questa relazione con alcuni aspetti che ci hanno dato soddisfazione. Riteniamo che la grafica sia in armonia con lo scopo dell'app: giocosa e leggera, ma allo stesso tempo semplice e intuitiva. Infine, l'obiettivo era creare uno strumento che semplificasse l'organizzarsi con gli amici per un pomeriggio di gioco: l'unione dei mezzi che forniamo, dalla creazione del match alla chat per organizzarsi, secondo noi può rendere l'app più funzionale di una semplice app di messaggistica.