RELAZIONE DEL PROGETTO TakeMyPet



**A.Accorinti, A.Di Carmine, G.Nesci**

# INDICE

[**INTRODUZIONE**](#_yspy8tt3f0xe) **2**

[**SOFTWARE DEVELOPER LIFE CYCLE**](#_g7km61aq0wuq) **3**

[**SOFTWARE UTILIZZATI**](#_8unclc6929dp) **3**

[**LINGUAGGI UTILIZZATI PER LA PROGETTAZIONE**](#_lquiyrwpy6ke) **3**

[**ANALISI DEI REQUISITI**](#_turkfg3b6wpx) **4**

[**CASI D’USO**](#_g26z4oqwyia0) **5**

[**DESCRIZIONE DEI CASI D’USO**](#_k7kmv08bwqm9) **7**

[**ENTITY RELATIONSHIP**](#_ujdsxceeujiz) **9**

[**WIREFRAME**](#_apisd699u5ci) **11**

[**CLASS DIAGRAM**](#_upsdn5xevax7) **15**

[**WEB APP**](#_vzedmcu3mu52) **17**

[**APPLICAZIONE ANDROID**](#_1iy4pj66xgkn) **22**

[**FIREBASE**](#_2nsbmrlo23dx) **25**

[**RICHIESTE WEB**](#_kwhwb1767dvy) **28**

[**SVILUPPI FUTURI**](#_sapqwj17twl5) **32**

# 

# 

# 

# INTRODUZIONE

Lo scopo di TakeMyPet è mettere in contatto in modo semplice ed intuitivo, i possesori di animali, con poco tempo da dedicare alla loro cura, con appassionati con del tempo libero, ma senza le risorse economiche per avere un animale. Non è un rapporto lavorativo, ma uno scambio di favori che avviene in modo gratuito(questo ci differenzia da altre app già esistenti).

Inoltre permette di creare e partecipare ad eventi, dando così l’opportunità di conoscere altre persone con la stessa passione.

È possibile iscriversi gratuitamente al servizio, il quale permette al pet sitter di accudire l’animale che preferisce tra quelli proposti dai proprietari vicini a lui.  
Viceversa i proprietari hanno la possibilità di scegliere, tra i pet sitters che hanno dato la loro disponibilità, quello che li convince di più in modo tale da poter lasciare il proprio " in mani sicure.  
La scelta è intuitiva ed avviene attraverso lo “swap” di profili proposti dal sistema in base ai parametri inseriti.

Avvenuto il match gli utenti vengono messi in contatto e possono incominciare il loro scambio di favori.

# SOFTWARE DEVELOPER LIFE CYCLE

Per lo sviluppo del nostro software ci siamo basati sul waterfall model, seguendo i seguenti passaggi:

# 

# 

# SOFTWARE UTILIZZATI

* ECLIPSE, VISUAL STUDIO CODE, ANDROID STUDIO, TORTOISE GIT
* MYSQL, HEIDISQL
* STARUML, GIMP, BALSAMIQ, POWERPOINT

# LINGUAGGI UTILIZZATI PER LA PROGETTAZIONE

* JAVA: Servlet, JPA, Hibernate, Maven, Jetty, Jackson, JUnit, Logback, Javax
* JAVASCRIPT: JQuery, Ajax
* ANDROID: Retrofit, Fragment, SQLite
* HTML, XML, CSS
* SQL

# ANALISI DEI REQUISITI

Al momento dell’iscrizione è possibile scegliere tra i due tipi di utente.  
Una volta effettuato l’accesso l’app cambierà la sua interfaccia e le sue funzionalità a seconda del tipo di utente.  
È presente anche un sito web, in cui l’admin può effettuare l’accesso e lavorare sulla propria dashboard di controllo dell’app.  
Inoltre, per gli utenti, nel sito è possibile registrarsi, effettuare l’accesso e modificare il proprio profilo.  
L’app deve avere le seguenti funzioni:

* ANNUNCI: crea annunci per fare accudire i tuoi animali (Proprietario).
* SWAP: scegli tra gli annunci disponibili (Pet Sitter) o tra i Pet Sitter che hanno accettato il tuo annuncio (Proprietario).
* PREFERITI: gli annunci o i Pet Sitter scelti nello swap vengono messi in preferiti, in modo da rivedere le scelte fatte e confermare con calma.
* EVENTI: crea e partecipa ad eventi (ad esempio “raduno per bassotti a Villa Villetta”), per conoscere altre persone con la tua passione.
* CHAT: trovato il match tra Proprietario e Pet Sitter i due utenti vengono messi in contatto nell’apposita chat.

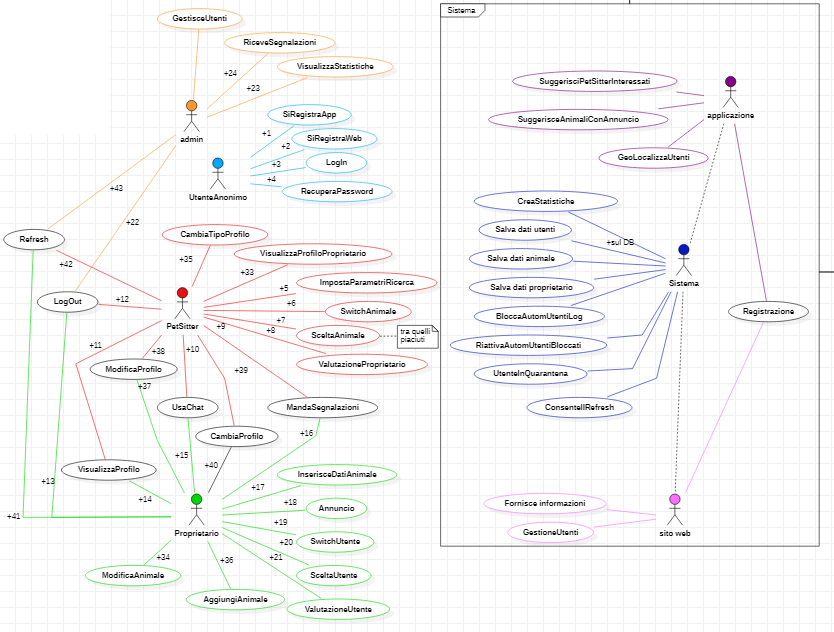
# 

# CASI D’USO

Abbiamo individuato in primo luogo gli attori del progetto, ovvero tutte le parti che hanno delle interazioni con il sistema.  
Tramite gli attori abbiamo stabilito le funzioni che possono compiere, utilizzando frasi e verbi.

Attori principali:

* PROPRIETARIO: principalmente inserisce annunci per i propri animali, sceglie i pet sitter, crea e partecipa ad eventi,invia valutazioni e segnalazioni.
* PETSITTER: principalmente sceglie gli annunci,crea e partecipa ad eventi,invia valutazioni e segnalazioni.
* ADMIN: gestisce le segnalazioni, modera gli utenti e gli eventi, visualizza le statistiche dell’applicazione.



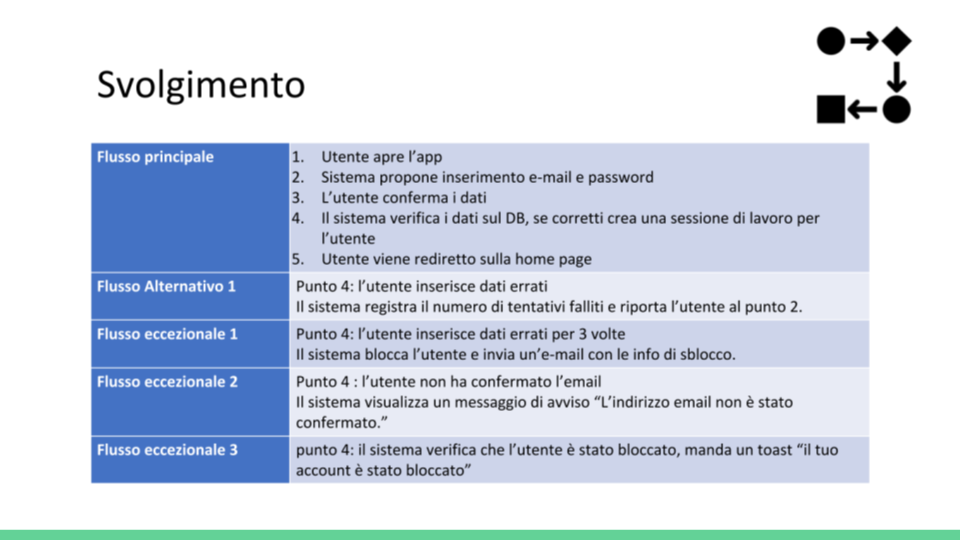
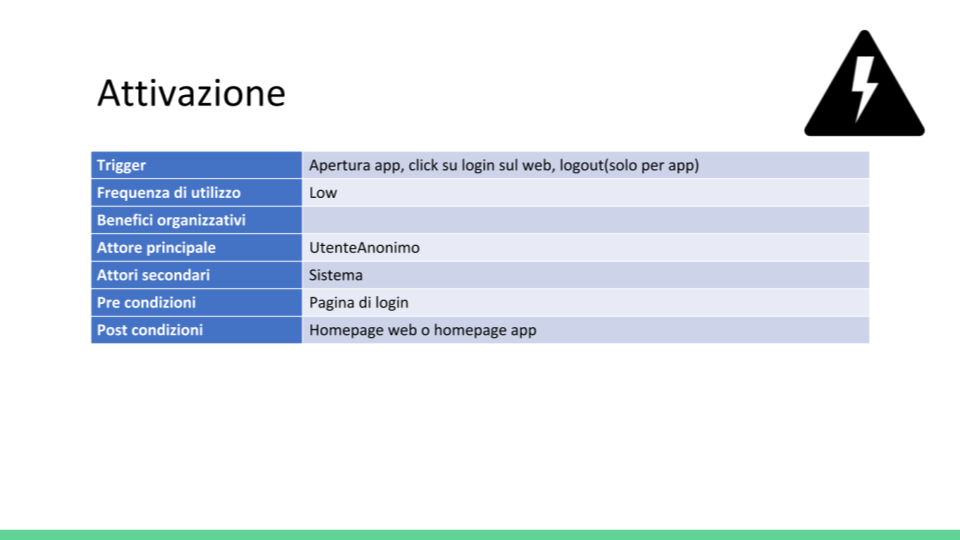
# 

# 

# DESCRIZIONE DEI CASI D’USO

Per ogni caso d’uso abbiamo provato a descrivere il modo in cui la nostra app dovrebbe funzionare indicando le informazioni generali,attivazione,svolgimento, flusso principale e flussi alternativi ([tutte le descrizioni sono disponibili qui)](https://drive.google.com/drive/u/4/folders/1y0ckG4XPYCL2vidUJClI52WHTY-V8Cgu).

Ad esempio ecco come abbiamo descritto il caso d’uso per il login.



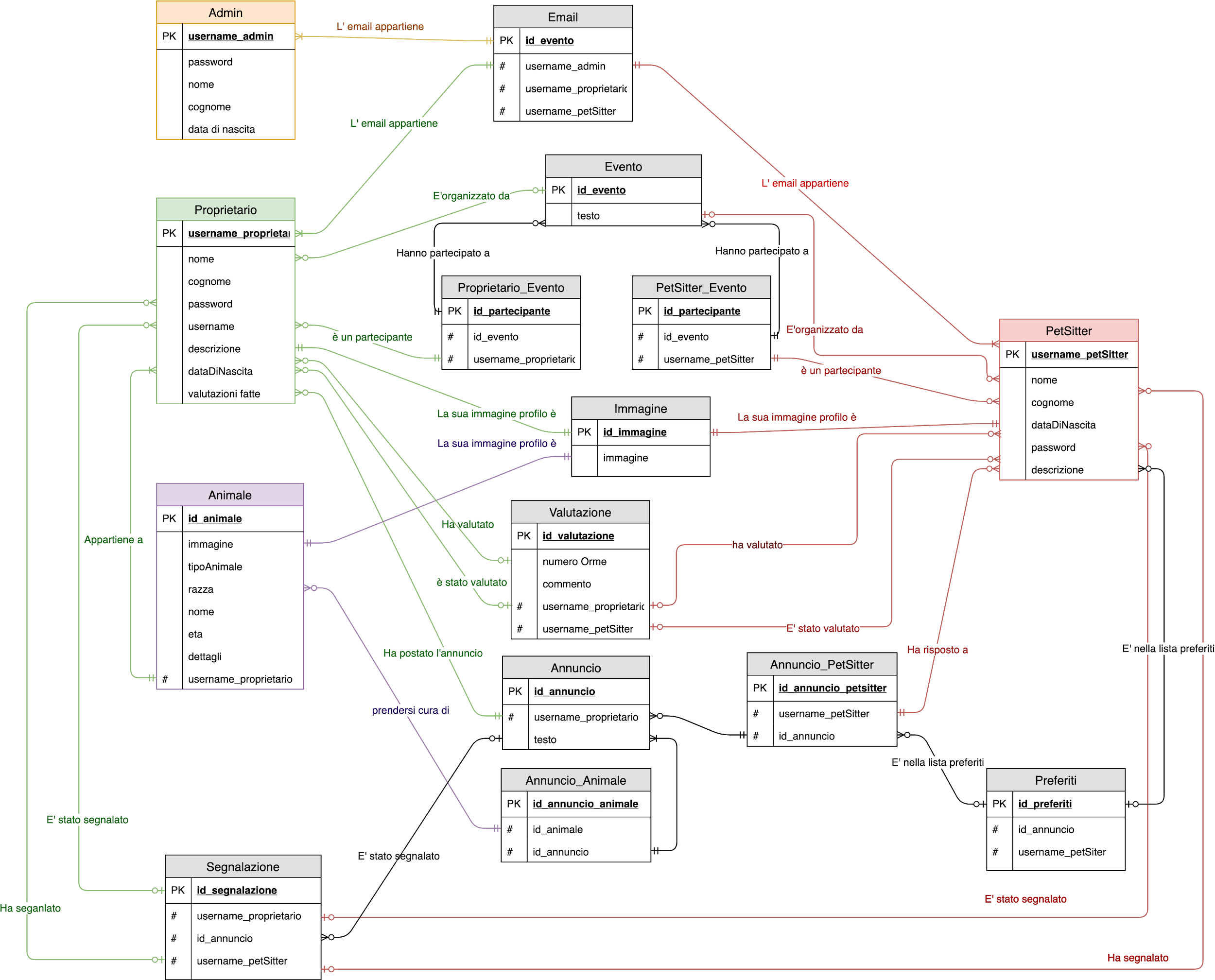
# 

# ENTITY RELATIONSHIP

Partendo dalle tre entità principali (Admin,Proprietario,Petsitter), abbiamo individuato i rispettivi attributi e abbiamo separato quelli che potevano rappresentare a loro volta un’altra entità.Successivamente, a partire dall’analisi dei casi d’uso, siamo andati a definire le entità che ci sarebbero servite per salvare i dati per il funzionamento dell’app (Preferito, Evento,Annuncio, Segnalazione, Valutazione).

Ad ogni entità abbiamo assegnando una chiave primaria.  
Una volta definite le entità abbiamo individuato le relazioni che le legano, con la loro cardinalità e stabilito chiavi secondarie e tabelle ponte.  
La tabella Animale ha una chiave secondaria che la mette in relazione con il Proprietario.  
La tabella Email è collegata agli utenti sempre con una chiave secondaria.  
Evento e Annuncio sono relazionate con le tabelle degli utenti tramite tabelle ponte.  
Segnalazione e Preferito hanno invece più chiavi secondarie che le relazionano agli utenti ed ognuna di esse esprime un attributo particolare di quella tabella (come ad esempio “preferitoDelPetSitter” , “preferitoDelProprietario” e “petSitterNelPreferitoDelProprietario”).

Per la gestione del database abbiamo usato la JPA ed utilizzato il relativo dialetto SQL per le interrogazioni.Alcuni esempi di query utilizzate nel progetto:

“SELECT c.dataRegistrazione , COUNT(c.username)AS dataReg FROM Utente c WHERE NOT c.tipoUtente = 'admin' GROUP BY c.dataRegistrazione ORDER BY c.dataRegistrazione”  
  
“SELECT c FROM Animale c WHERE c.proprietario= :username”  
  
“SELECT c FROM Evento c WHERE c.organizzatore= :username”  
  
  


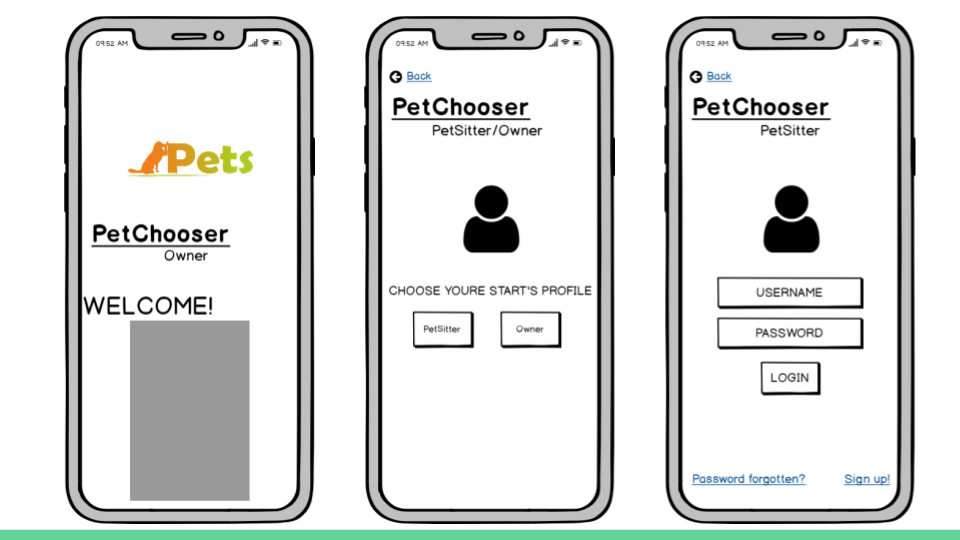
# 

# 

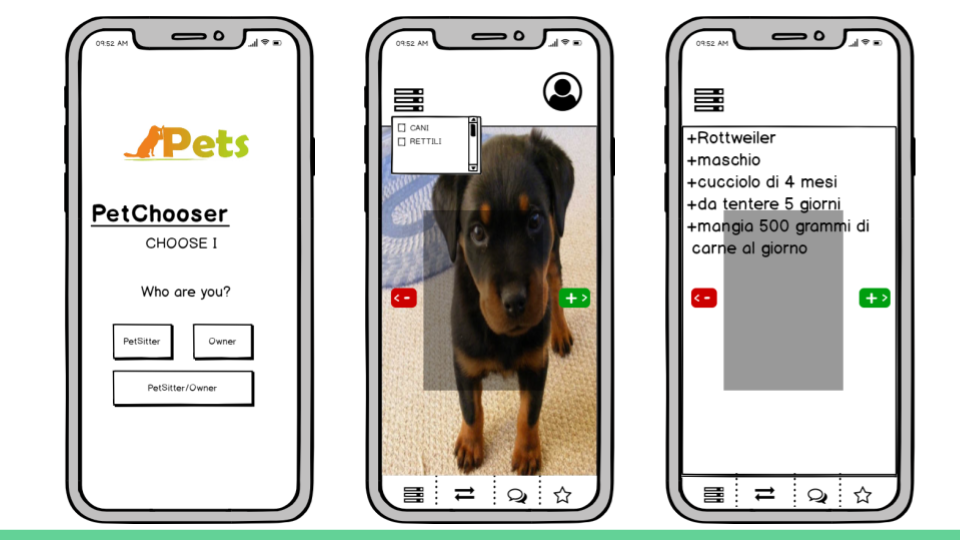
# WIREFRAME

Prima di passare alla fase di progettazione, abbiamo immaginato la “User Experience” e la “User Interface” attraverso i wireframe.

Abbiamo cercato di facilitare l’esperienza e l’utilizzo del software da parte dell’utente, utilizzando lo spazio a disposizione per rendere l’app più usabile ed intuitiva possibile.



Wireframes con la schermata iniziale e quella per il login con la possibilità di scegliere tra petsitter e proprietario(nel caso di doppio profilo).

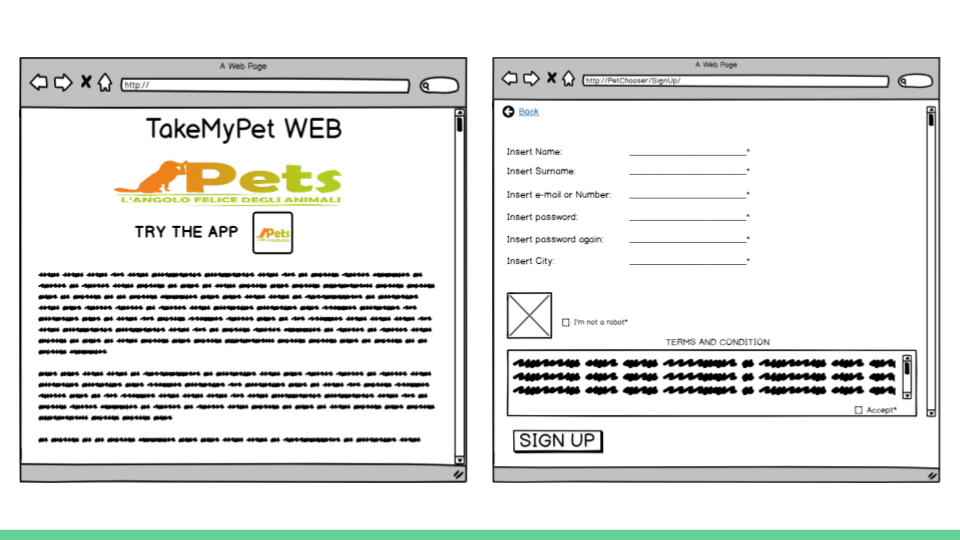


Wireframe con la funzione di “Swap” dell’annuncio con le caratteristiche dell’animale

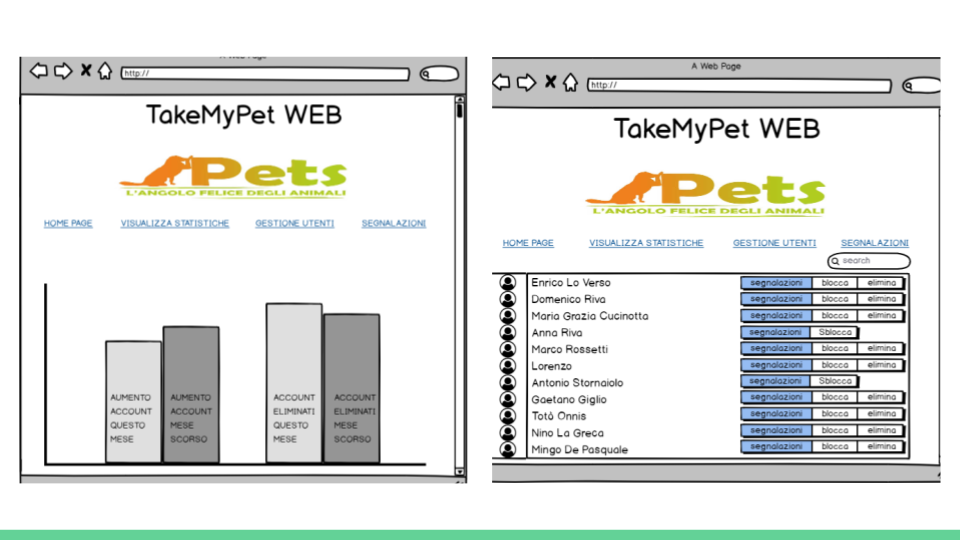
Wireframe con la funzione di ricerca e creazione dell’evento



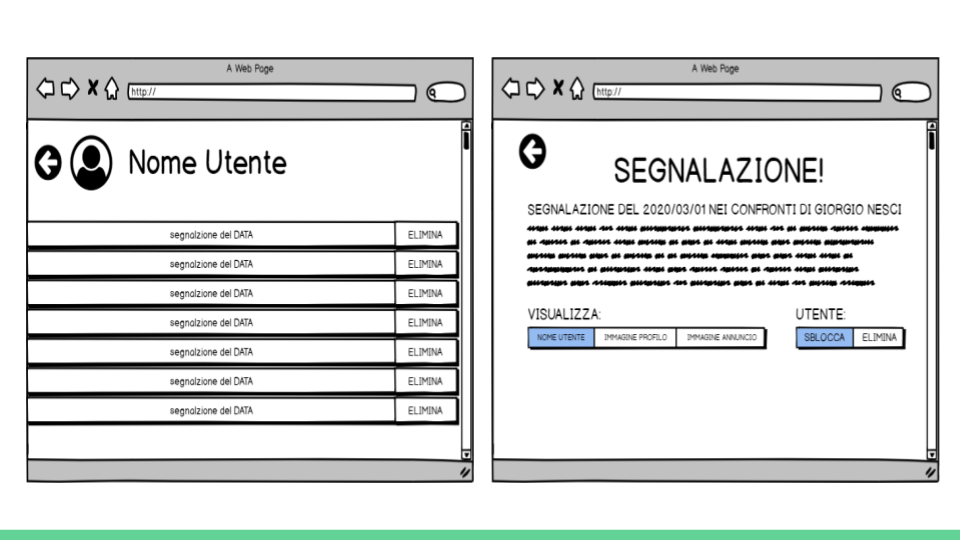
Wireframe con una la chat e la schermata preferiti.



Wireframe con la home e la schermata di registrazione del sito web.



Wireframe con le schermate di gestistione degli utenti e delle statistiche, visualizzabile dall’admin.



Wireframe per la gestione delle segnalazioni e la visualizzazione dei dettagli.

# CLASS DIAGRAM

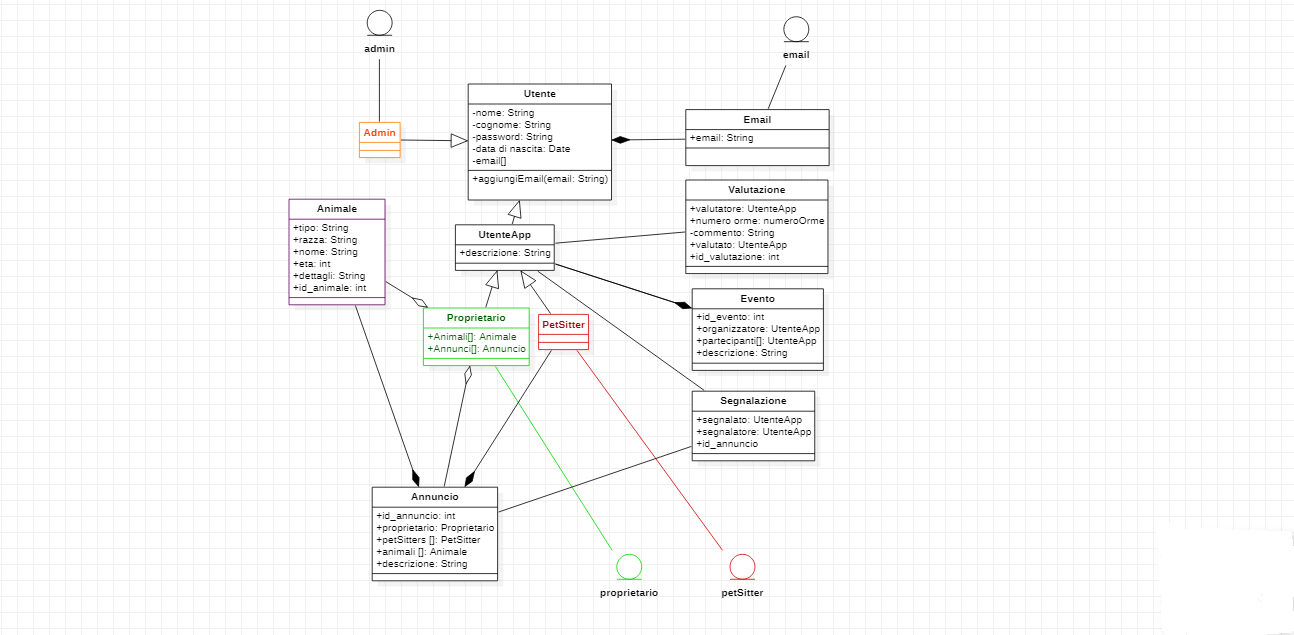
Nella fase di progettazione abbiamo applicato i design pattern fondamentali per una buona riuscita del progetto.

Per la comunicazione abbiamo seguito il modello MVC (model, view, controller).  
I model sono stati realizzati attraverso l’incapsulation e la creazione di java beans.

Le classi business, contenenti i metodi chiamati dai controller, sono state fatte a partire da interfacce stabilite in precedenza, creando una serie di “contratti” da rispettare (design by contract) .Abbiamo inoltre usato un singleton per l’entity manager della jpa (per evitare di occupare troppo spazio nella RAM) ed una factory per la creazione degli utenti.

Sia le business, che le interfacce e i model seguono un albero di ereditarietà che è stato strutturato astraendo e trovando degli elementi comuni nelle classi.

I controller sono stati creati con le servlet, che risponderanno alle varie view, gestite da jQuery (nel caso del web) e da Android (nel caso dell’app).

In tutto il progetto sono stati tenuti sempre bene a mente i 5 principi SOLID e più in generale il principio di keep it simple.

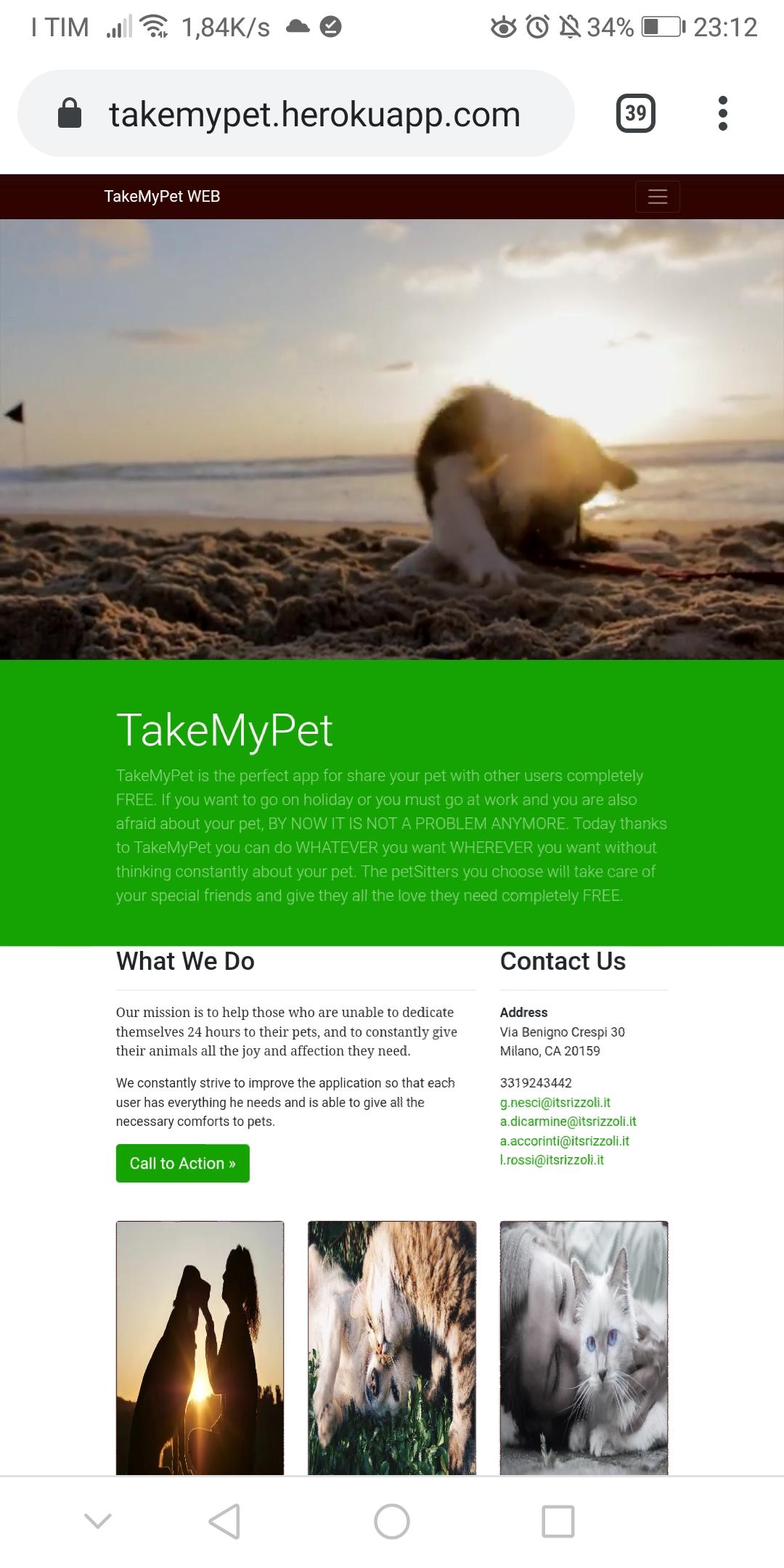
# 

# Nei model abbiamo definito una superclasse Utente che racchiude gli aspetti comuni per tutti gli utilizzatori della nostra applicazione. Questa classe viene estesa da Admin e da UtenteApp che a sua volta viene estesa dalle classi più specifiche Proprietario e PetSitter.I model degli utenti contengono al loro interno attributi che li relazionano (e creano chiavi secondarie nel database) ai model delle funzionalità dell’app e viceversa (ad esempio Annuncio,Animale,Segnalazione,Email).Le interfacce sono state create in base agli utenti (in quanto le funzionalità possono variare nel tempo). UtenteAppInterface ha al suo interno tutti i metodi astratti comuni ai due tipi di utente ed è estesta da ProprietarioInterface e PetSitterInterface, che contengono i metodi più specifici.Abbiamo creato anche una interfaccia per le funzioni dell’admin, per la registrazione e per il login. Tutte le interfacce vengono implementate dalle omonime classi business e le classi ProprietarioManager e PetSitterManager, estendono UtenteAppManager, in modo da non dover implementare gli stessi metodi comuni in due classi diverse.

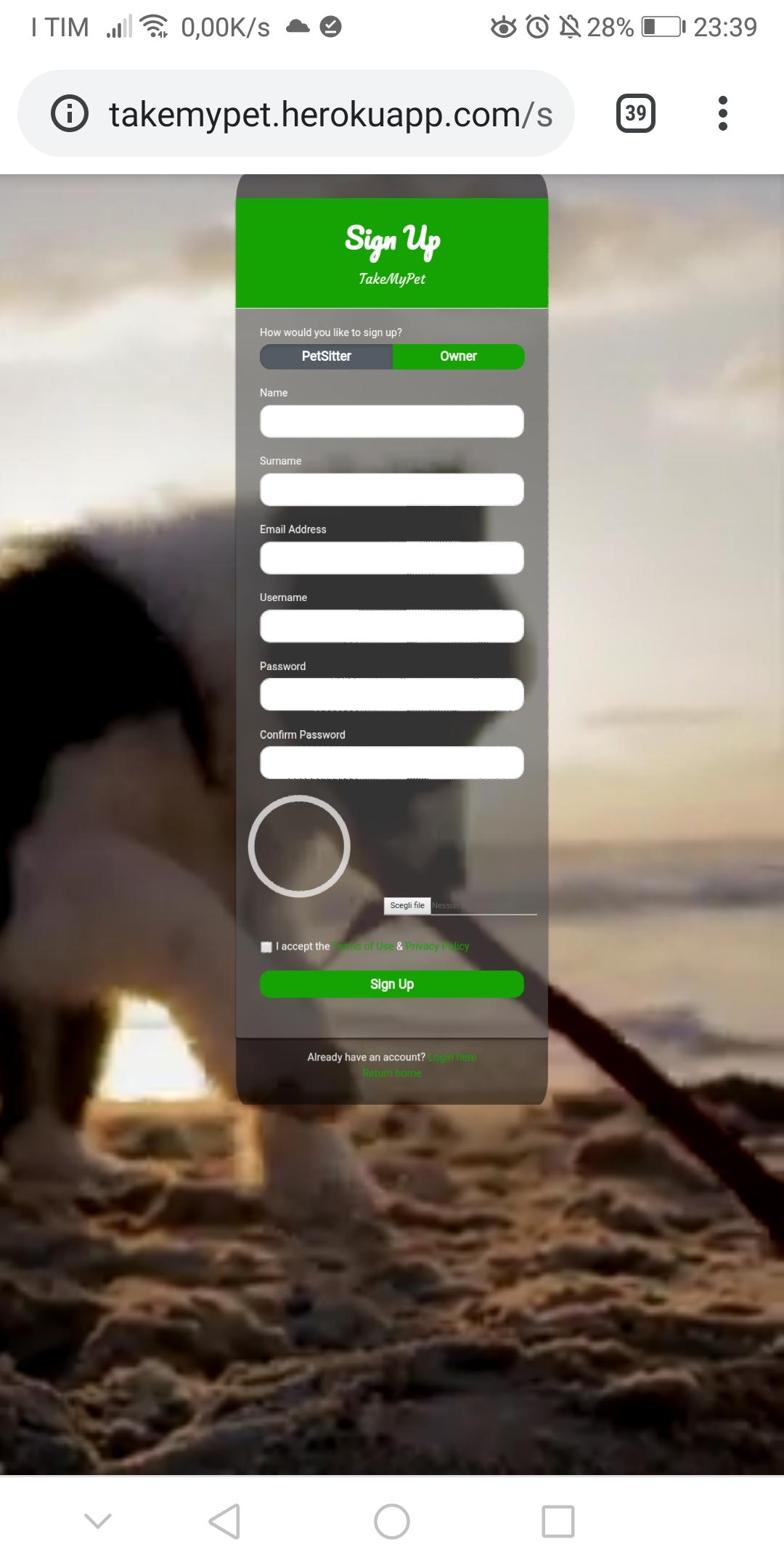
# 

# WEB APP

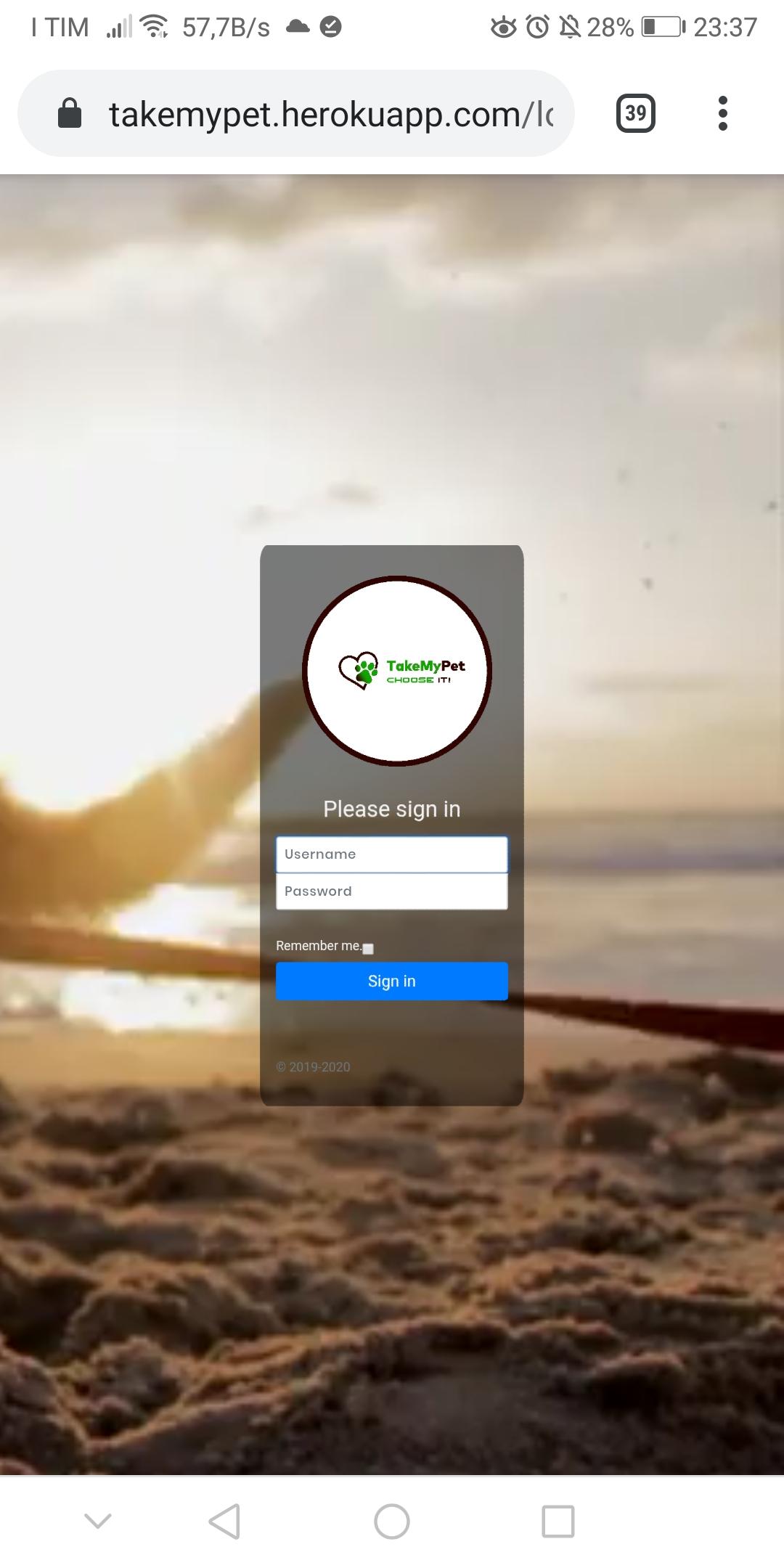
Il sito web è reso dinamico dall’utilizzo di Bootstrap 4 e JQuery. Bootstrap è una libreria opensource per la creazione di pagine web dinamiche che utilizza HTML, CSS, and JS.



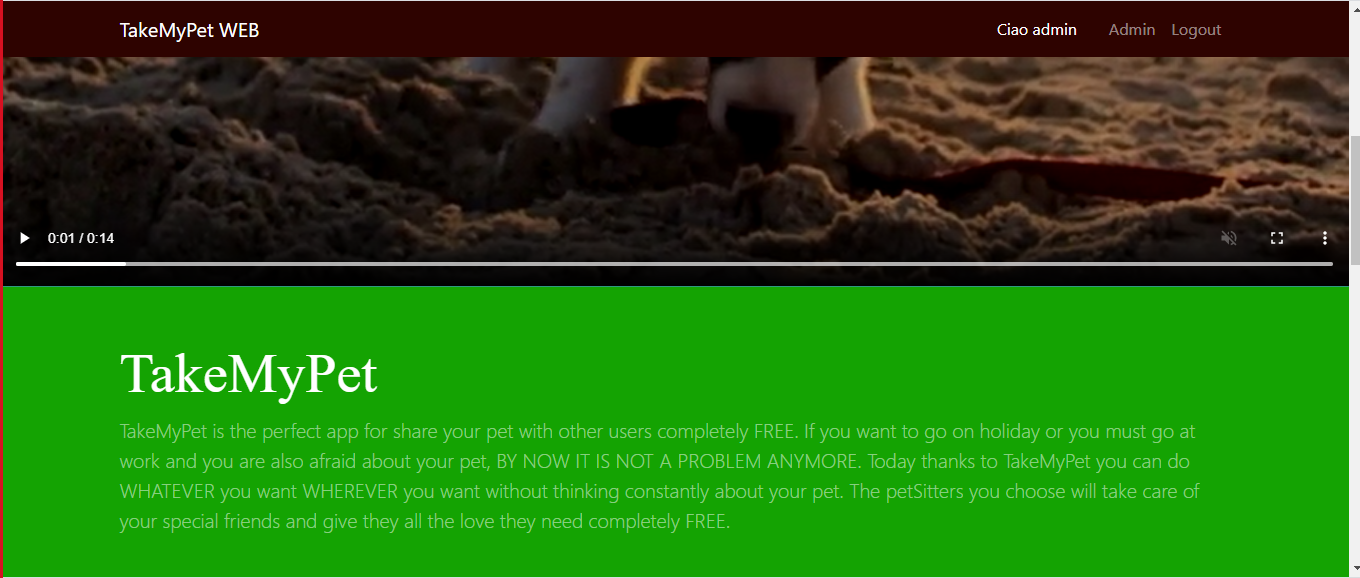
Dal sito, oltre a trovare informazioni sull’applicazione, ci si può registrare e scaricare l’app dal play store.  
È possibile inoltre effettuare il login, visualizzare il proprio profilo ed effettuare modifiche.



Nella registrazione la web app controlla che un utente con lo stesso username non esista già nel database, se i campi sono riempiti, se la mail inserita è nel formato giusto e controlla se la conferma della password sia corretta.  
Permette anche il caricamento dell’immagine del profilo: il Base64 dell’immagine viene inviato al server che lo converte in un bytearray, attraverso il quale crea una immagine .png che viene salvata lato server.   
La comunicazione avviene dalla pagina verso il server attraverso Ajax.  
Nel database viene salvato l’utente con i campi inseriti nel form e per l’immagine si salva solo l’url.



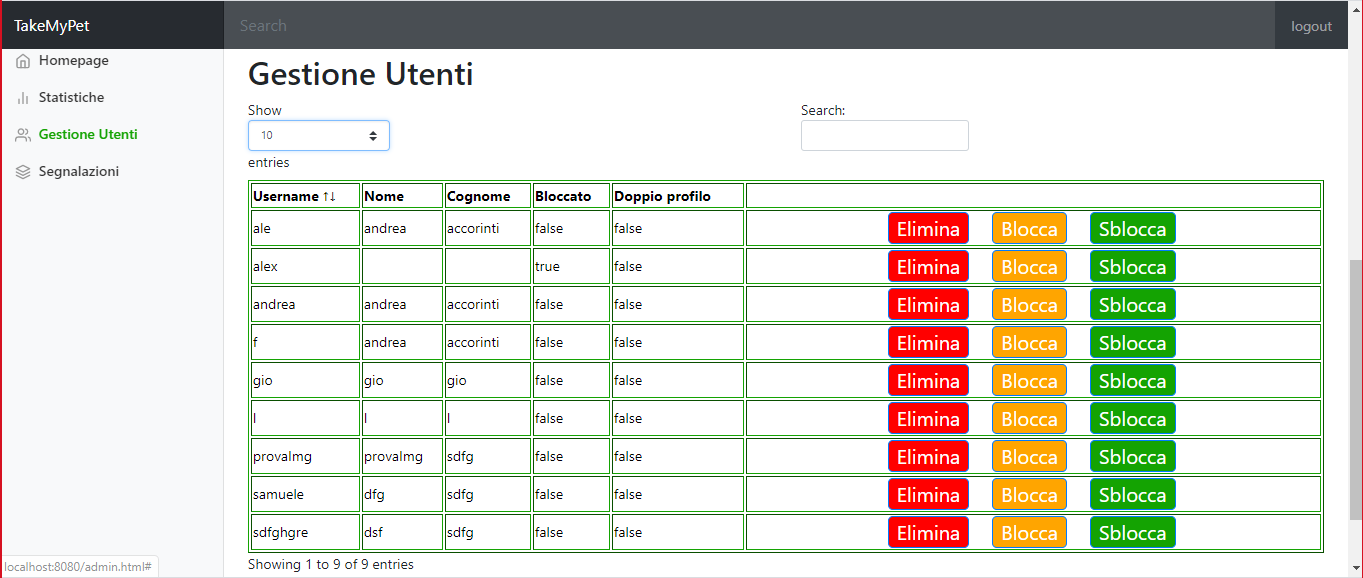
Nel login, cliccando su “sign in”, la pagina chiama il server che controlla che l’utente esista nel database e carica una pagina diversa a seconda che si tratti di un utente o di un admin (HomePage o Dashboard).   
La servlet risponde inviando un Json con i dati dell’utente (esclusa la password) che viene salvato nel session storage o nel local storage nel caso di spunta nel “remember me” e che servirà poi per la manipolazione del DOM.  
Se non trova l’utente o la password è sbagliata appare la scritta in rosso “username o password errati”.  
Se viene inserita per dieci volte una password sbagliata, l’account viene bloccato (cambiando un attributo sul database) e viene inviata all’utente una mail contenente il link per accedere alla pagina di sblocco (simile alla pagina di login) ed un codice alfanumerico (generato casualmente) da inserire per sbloccare l’account.



Una volta fatto il login l’admin può accedere alla dashboard di controllo dell’app.  
Attraverso la dashboard l’admin può visualizzare statistiche, gestire gli utenti e visualizzare le segnalazioni.



Nella home della dashboard è presente un grafico contenente le statistiche. La pagina chiama una servlet che fa una query nel database (in questo caso il numero di utenti iscritti ogni giorno).  
In base alla risposta si genera un grafico con ChartJs( una libreria per Javascript).

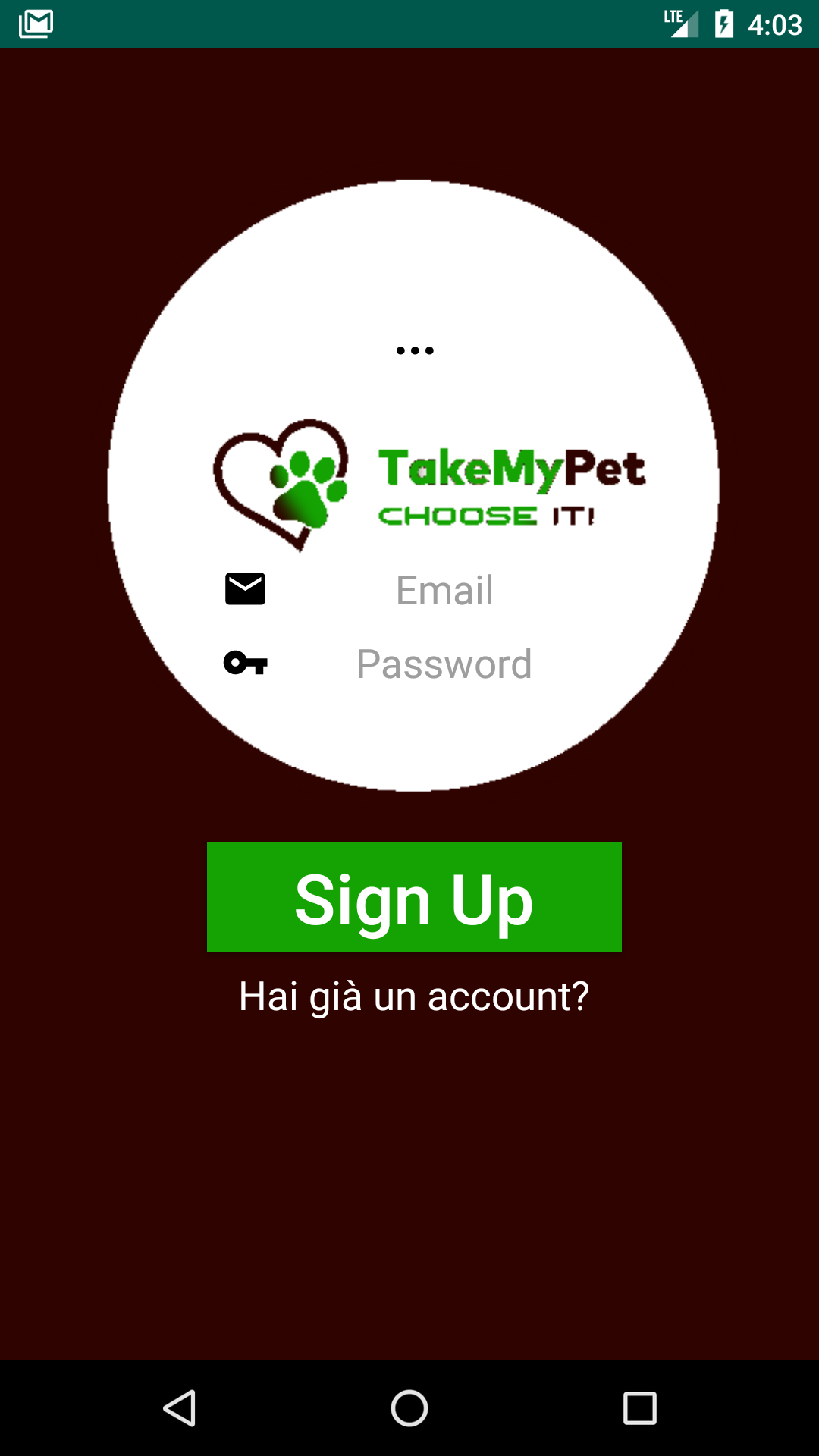


Attraverso la sezione “gestione utenti” l’admin visualizza un elenco con tutti gli utenti registrati.  
La tabella utilizza DataTableJS.  
In questa tabella si possono bloccare, sbloccare ed eliminare profili attraverso chiamate Ajax che comunicheranno al server di effettuare le modifiche sul DB.

# 

# APPLICAZIONE ANDROID

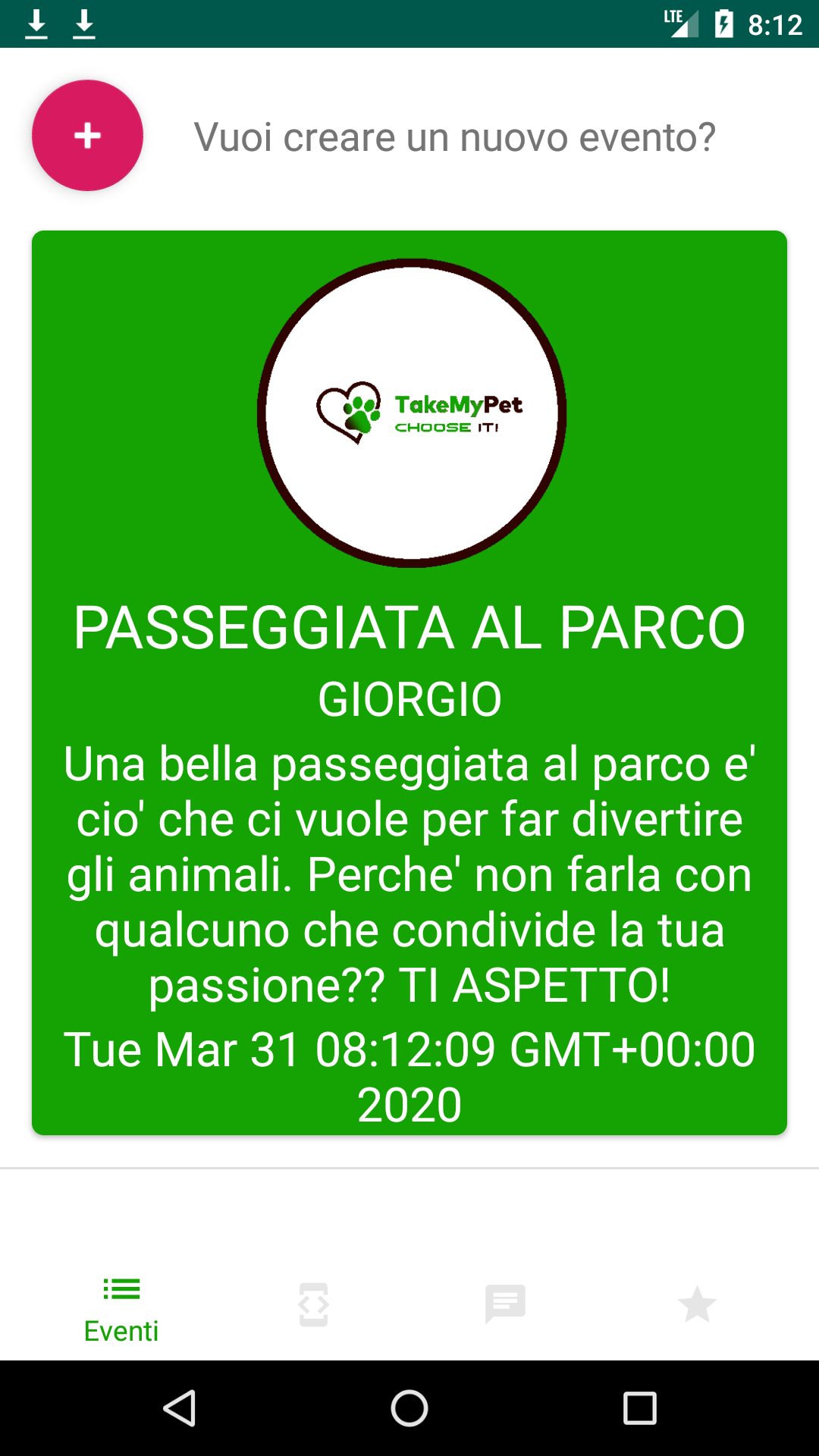
Per la comunicazione con il server abbiamo utilizzato la libreria Retrofit implementando una interfaccia per le richieste HTTP.   
Le chiamate avvengono in modo asincrono e sul dispositivo vengono salvati i dati essenziali per il funzionamento dell’app, utilizzando SQLite.  
Le schermate dell’interfaccia utente vengono gestite dinamicamente utilizzando i Fragments, che si intercambiano in una activity.



L’utente può registrarsi anche attraverso l’app.  
Al momento della registrazione può specificare se registrarsi come PetSitter o come Proprietario.  
La servlet chiamata per la registrazione è la stessa che viene usata nel web, e si comporta allo stesso modo: facendo le dovute verifiche e registrando l’utente nel database (compreso per l’immagine del profilo, per il blocco dell’account e per l’invio dell’email per lo sblocco).

# 

L’utente deve loggarsi per utilizzare l’applicazione.   
Il funzionamento del login avviene in modo analogo al sito web, ma utilizzando Retrofit.   
Il server risponderà con un Json contenente i dati dell’utente, esclusa la password.  
Una volta loggato l’utente può navigare attraverso una AppBar posta in basso, spostandosi da una schermata ad un’altra attraverso il click o il semplice slide touch sullo schermo.

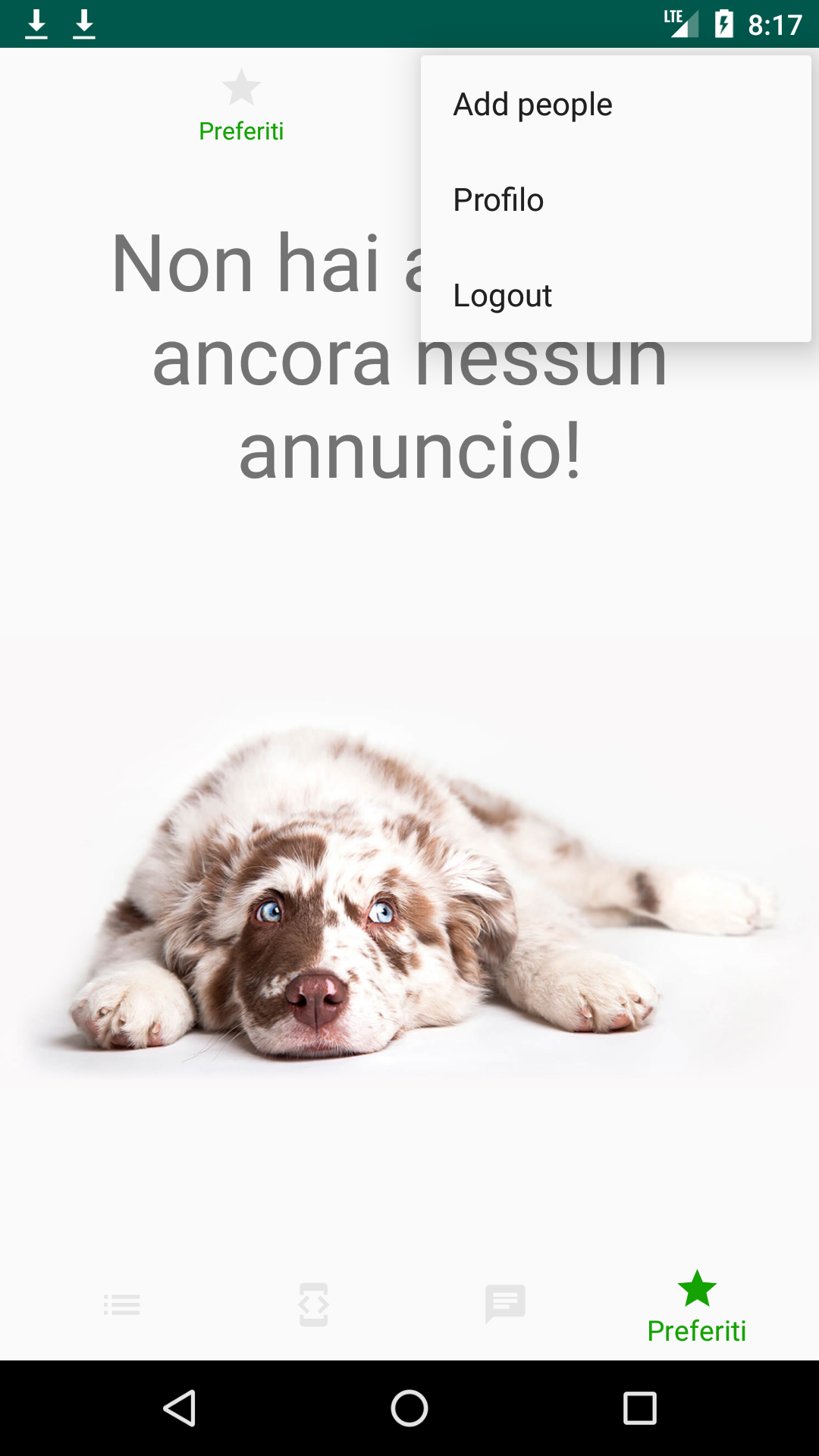


Sulla schermata eventi è possibile visualizzare quelli già esistenti ,controllare lo stato dei propri, crearne di nuovi e partecipare a quelli creati dagli altri (attraverso servlet specifiche per ogni tipo di utente).

In una fase più avanzata dello sviluppo, in questa schermata (attraverso le query della servlet che risponde alla chiamata), verranno caricati gli eventi in ordine di vicinanza all’utente (tramite geolocalizzazione).   
Gli eventi creati dall’utente e quelli a cui partecipa, saranno visualizzabili attraverso la selezione da un menù a tendina.



l Proprietario può creare annunci per i propri animali ed usare la funzione di “swap” per trovare il petsitter più adatto, tra quelli che hanno dato la conferma.  
Quando il Proprietario crea un annuncio, può selezionare uno o più animali dalla sua lista animali, aggiungere una descrizione, una data e la posizione.   
L’annuncio viene salvato nel DB in una apposita tabella con le dovute chiavi secondarie.  
Gli annunci creati vengono visualizzati dal Proprietario in questa schermata, selezionando un annuncio verrà attivata la modalità swap in cui può selezionare i PetSitter a lui graditi e inserirli nella lista preferiti.   
Chiamando il server ogni volta che si seleziona l’annuncio, esso sarà sempre aggiornato con i nuovi PetSitter interessati.  
Usando lo “swap” il PetSitter può trovare l’animale che preferisce e aggiungerlo alla lista dei preferiti.



In questa schermata i PetSitters accettano o eliminano gli annunci salvati in preferiti.  
Dopo che un annuncio è stato accettato da un Petsitter, esso apparirà nello swap del Proprietario corrispondente a quell'annuncio.  
il Proprietario può rivedere ed eliminare i PetSitters salvati in preferiti tramite lo swap e, dando la conferma ad uno di essi, i due verranno messi in contatto.



L’utente inoltre può modificare le sue informazioni personali direttamente dall’applicazione.  
I dati verranno aggiornati nel database lato server e in quello locale.

# FIREBASE

Nella demo realizzata nella presentazione per il cliente, l’app utilizza Firebase (un database NoSQL).

Firebase infatti consente di sostituire la parte di realizzazione del Backend fornendo un'interazione molto facile e veloce. Tra i punti di forza vi sono:

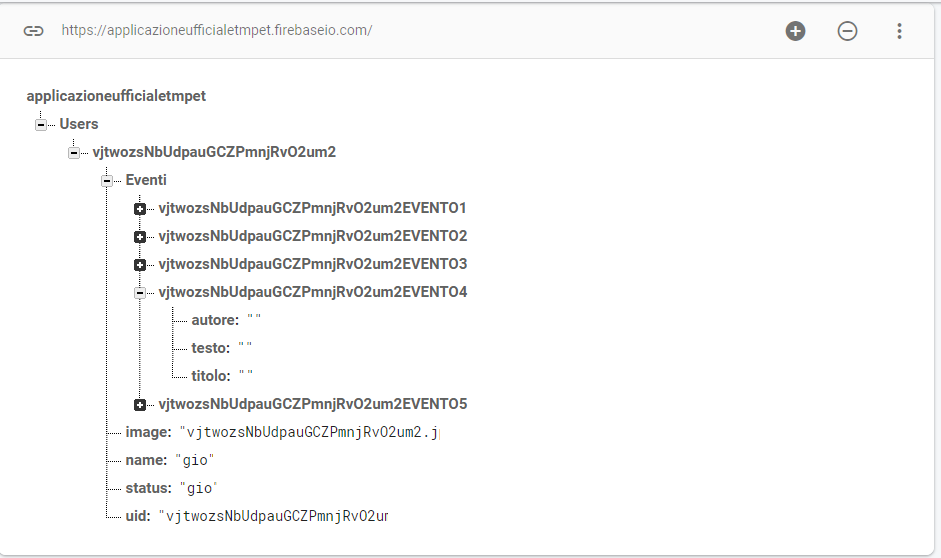
-sincronizzazione dei dati in maniera istantanea ogni qualvolta il dispositivo è connesso;

-disponibilità delle librerie client per integrare Firebase in ogni tipo di app;

-sicurezza: i dati immagazzinati in Firebase sono continuamente sottoposti a backup. La comunicazione con i client avviene sempre in modalità crittografata.

Firebase ci ha consentito di implementare rapidamente la registrazione e il login per la nostra applicazione utilizzando alcuni metodi consigliati come “.createUserWithEmailAndPassword(mail, password)” e “.signInWithEmailAndPassword(mail, password)”, ci ha consentito di memorizzare i dati dell’utente come ad esempio username e stato, e ci ha permesso di salvare l’immagine profilo dell’utente.

Per ogni utente siamo riusciti a memorizzare tutti i dati riguardanti gli eventi creati con un UID univoco in modo da poter implementare, grazie anche all’utilizzo della geolocalizzazione, la possibilità per un utente di visualizzare tutti gli eventi vicini a Lui.

In Firebase non vi sono né tabelle né record, ma tutti i dati inseriti vanno a costituire un albero JSON.

# 

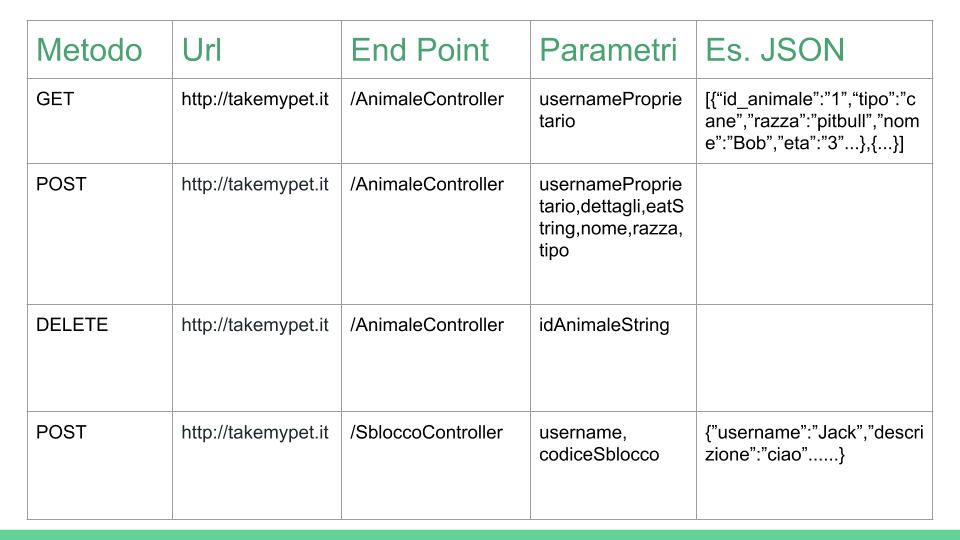
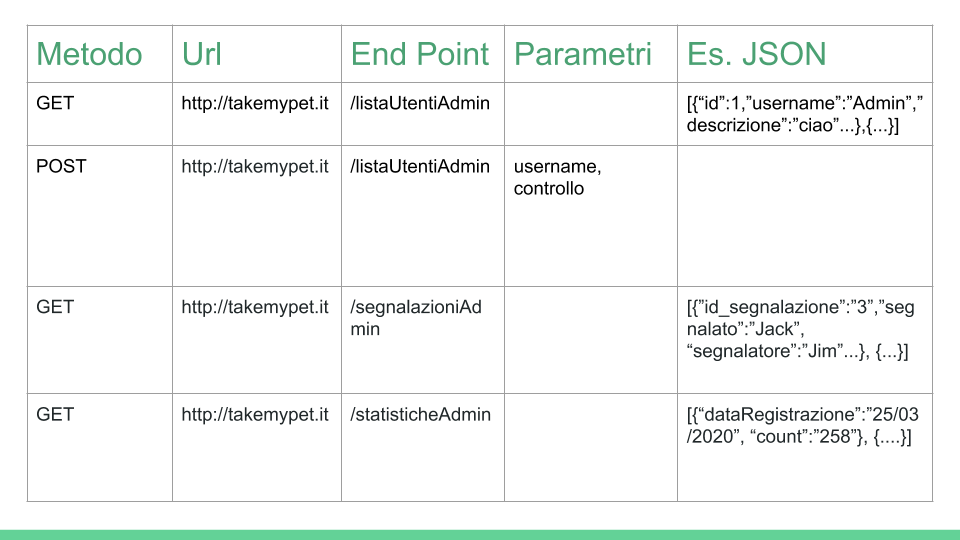
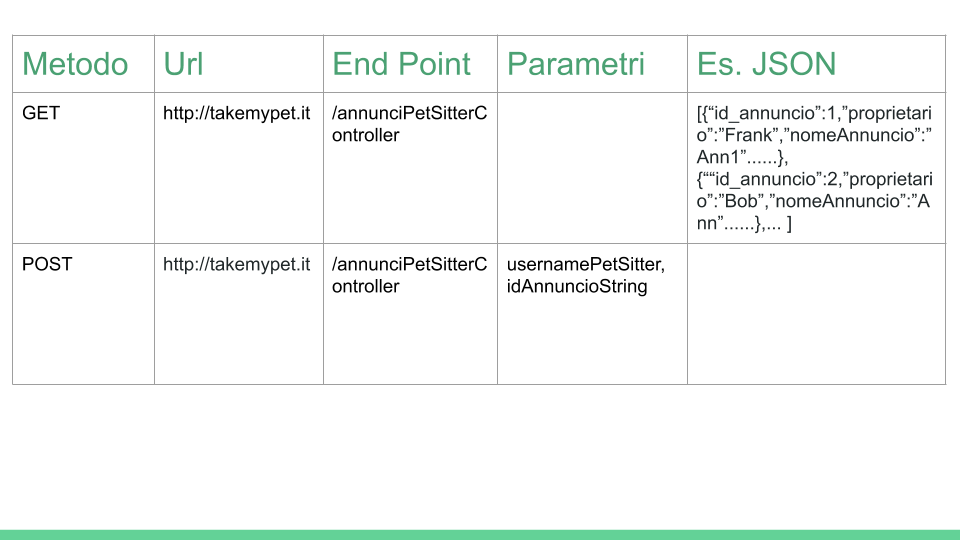
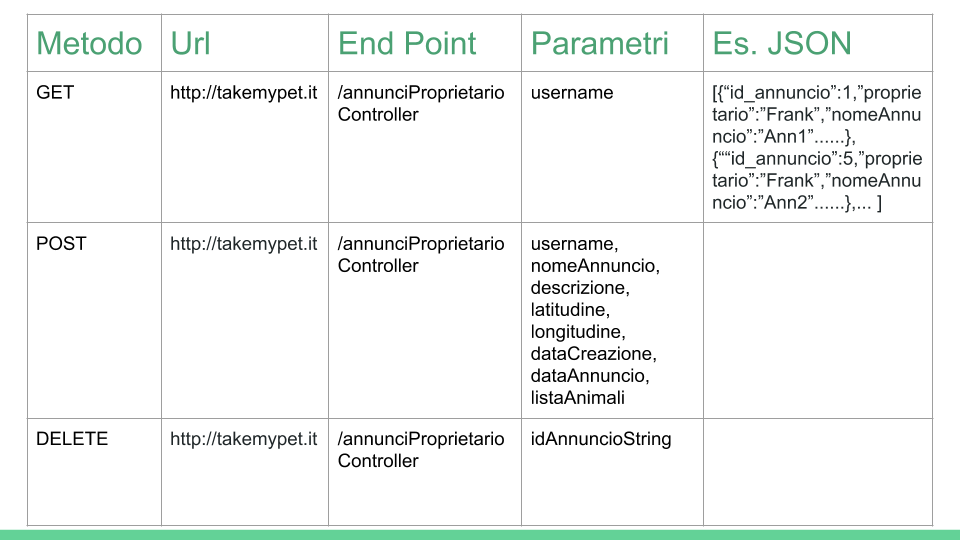
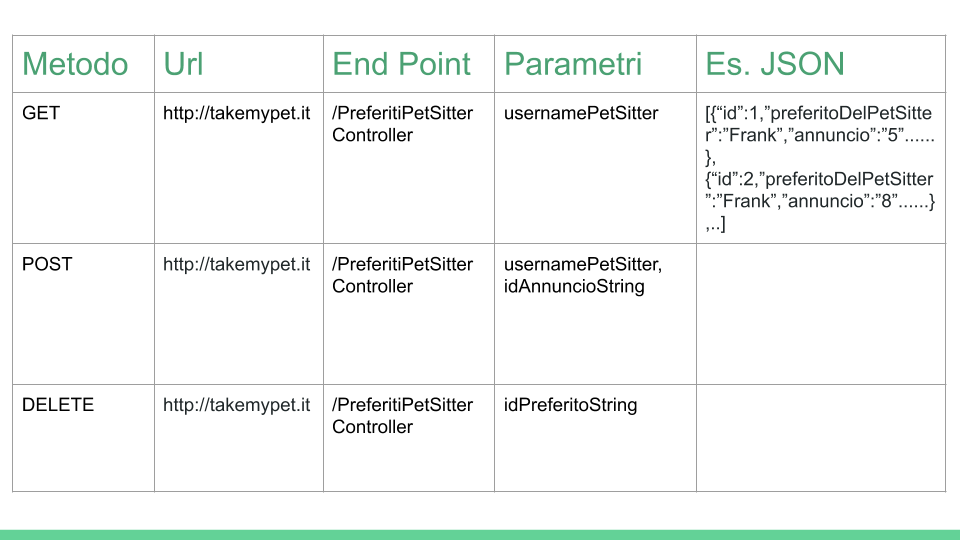
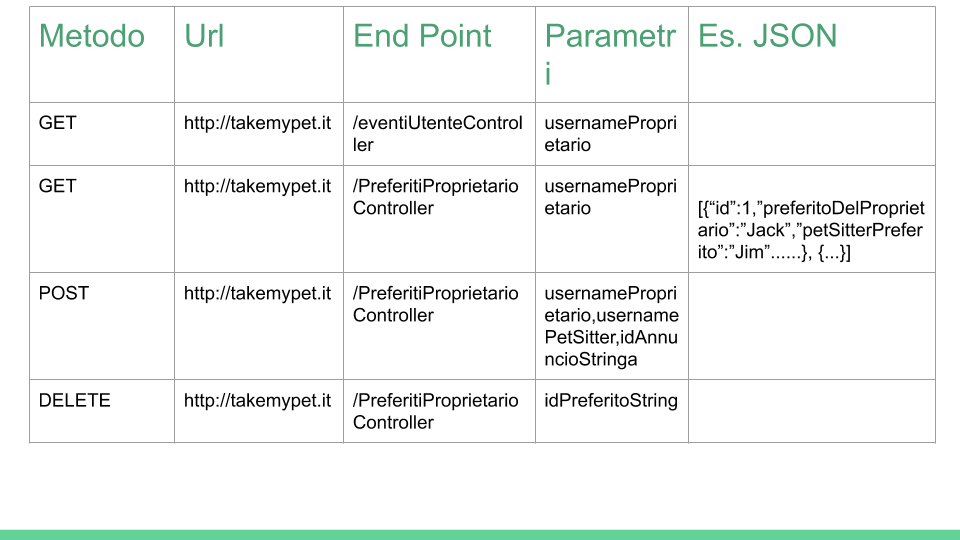
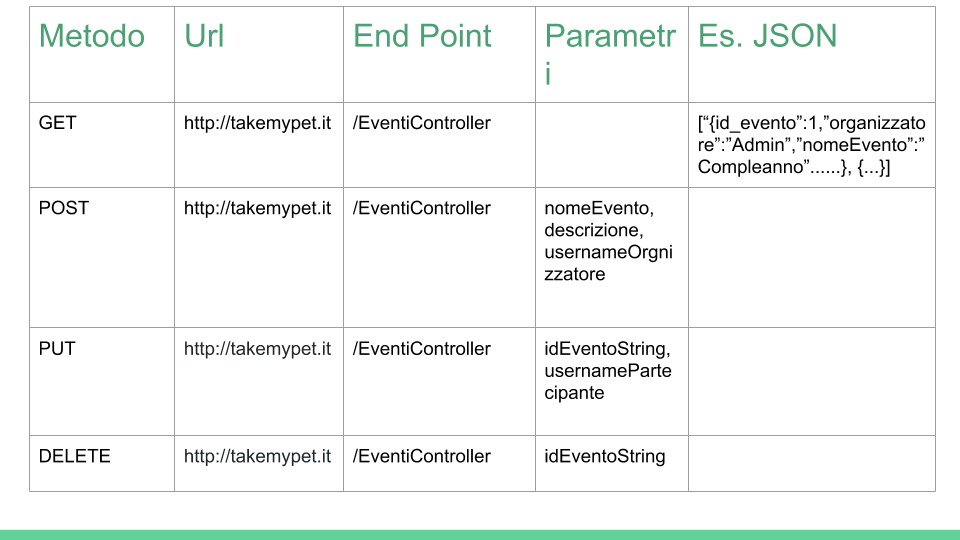
# 

# 

# 

# 

# RICHIESTE WEB



# 

# SVILUPPI FUTURI

Aggiungere la geolocalizzazione

Aggiungere la funzione di chat

Aggiungere la funzione doppio profilo  
Aggiungere funzione di recupero password