POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica (Computer Engineering)

Tesi di Laurea

Sviluppo di una piattaforma mobile per favorire le adozioni di animali domestici

Relatore
Prof.re Fiori Alessandro

Candidato
Carlo Cafaro Macchi
matricola: 270717

Anno Accademico 2022-2023

Indice

Elenco delle figure

Sommario

Lo scopo di questa tesi è lo sviluppo di una piattaforma mobile volta al facilitare i processi di adozione di animali domesitci situati nei canili o strutture provvisorie come rifugi, ma soprattutto di ridurre l'abbandono di tali animali. La piattaforma è in grado di far registrare sia gli utenti che sono interessati all'adozione che i responsabili e/o dipendenti delle associazioni e canili che prenderanno parte al progetto. L'applicazione mobile é stata realizzata mediante il frameowrk Flutter, basato sul linguaggio Dart. Diversamente, il backend è stato realizzato con il web framework Django, seguendo il paradigma Model-View-Template. La funzionalità cardine del progetto, volta a ridurre il tasso di abbandono, è il sistema di match tra utente e animale domestico realizzato con un recommendation system basato sulla similarità del coseno.

Capitolo 1

Introduzione

1.1 Il fenomeno dell'abbandono degli animali

Ogni anno con l'arrivo dell'estate, purtroppo si sente parlare spesso di quello che viene chiamato il fenomeno dell'abbandono degli animali domestici. Dal 2020 tuttavia le adozioni di animali domestici soprattutto nei canili e rifugi, sono aumentate notevolmente non abbastanza però da fermare l'abbandono. I motivi sono i più svariati: chi non può occuparsi del proprio animale domestico oppure chi non riesce ad organizzare un viaggio adatto ai propri amici a quattro zampe. Si stima che ogni anno vengano abbandonati circa 50.000 cani e 80.000 gatti domestici e soprattutto in estate, stagione nella quale si registra il 30% dei casi di abbandono [1]. L'abbandono è punibile per legge con arresto e ammenda come enuncia l'articolo 727 del Codice Penale, pertanto il problema è stato ridotto ma non eliminato. Anche l'avvento dei microchip, obbligatori per legge nei cani e facoltativi nei gatti, ha contribuito alla riduzione del tasso di abbandono, ma non è stato comunque sufficiente a frenare o ridurre i numeri vertiginosi.
Quando ci si imbatte in un animale domestico abbandonato è buona pratica seguire i consigli dell'**ENPA** (Ente Nazionale Protezione Animali) tra cui non fermare o rincorrere un animale abbandonato in autostrada. Se in città invece mettere le quattro frecce, avvicinarsi piano e metterlo in sicurezza offrendogli cibo o acqua. Infine è sempre consigliato chiamare il 112 segnalando la posizione dell'animale abbandonato e in seguito individuare e contattare la sede ENPA più vicina.

Inoltre da qualche anno è stata lanciata la campagna "#amamiebasta"[1] dall'**Anas** e dalla **LNDC** (Lega Nazionale Del Cane) volta a scoraggiare i proprietari ad abbandonare i loro animali domestici prima di andare in vacanza denunciando l'atto alle forze dell'ordine o all'Anas e di segnalare la targa del veicolo se si assiste in diretta.

1.2 Obiettivi e Struttura della tesi

Lo scopo principale di questa tesi è la riduzione del tasso di abbandono di animali domestici, andando ad unirsi a tutte le campagne e consigli già esistenti. Per tale motivo, è stata sviluppata un'applicazione mobile a tutto tondo (mobile client, backend e database) per favorire le adozioni di tutti quegli animali che si trovano in canili, rifugi o associazioni. Tale obiettivo vuole essere raggiunto informando preventivamente, tramite la piattaforma, tutte le informazioni circa taglia, razza e età dell'animale andando così a ridurre i ripensamenti e/o gli eventuali abbandoni. Inoltre cardine del progetto è il sistema di matching tra utente finale e animale domestico in grado di valutare l'affinità producendo una percentuale indicativa di tale valore.
La tesi è stata divisa in capitoli ognuno dei quali verte su una tematica specifica del progetto. Il capitolo 2 espone e fa una carrellata dei Recommendation Systems, tecnica usata per abilitare il match tra utente e animale domestico. Nella parte finale del capitolo vengono presentati i principali concorrenti già in mercato alla data di stesura di questo documento. Nel capitolo 3 vengono analizzate tutte le tecnologie coinvolte nello sviluppo della piattaforma mobile realizzata. Diviso in due sezioni, la prima verte sulle tecnologie impiegate per sviluppare un front-end mobile e nello specifico Flutter. Nella seconda sezione del capitolo invece viene ampliato il framework Django usato per realizzare il backend. Per ultimo vengono trattate le varie tipologie di database. Ogni tecnologia analizzata viene presentata insieme a delle alternative in modo tale da permettere al lettore di avere una conoscenza sia generale che specifica. Il capitolo 4 e il capitolo 5 sono specifici della piattaforma sviluppata e si concentrano rispettivamente sull'architettura (passando per analisi funzionali e modelli dei dati) e sui casi d'uso ovvero una sorta di manuale d'utilizzo dell'applicazione mobile. Nel capitolo 6 vengono espresse le conclusioni e proposte nuove migliorie da aggiungere ad un'eventuale versione futura del progetto.

[1] https://www.stradeanas.it/it/lazienda/chi-siamo/le-nostre-iniziative/amamiebasta

Capitolo 2

Stato dell'arte

2.1 Recommendation Systems

Nell'ultimo ventennio, con l'avvento di Internet accessibile dalla maggiorparte della popolazione, la nascita e crescita dell'e-commerce (Amazon), il boom dei social media (Instagram) e il passaggio allo streaming di film e musica (Netflix, Spotify), sono anche nati i sistemi di raccomandazione o **recommendation systems**. Si tratta di motori software alimentati da Big Data o intelligenza artificiale che suggeriscono o raccomandano un certo tipo di prodotto al consumatore [2]. É chiaro che questi sistemi siano fonte di grandi guadagni per le aziende che sfruttano il loro potenziale poichè guidano il consumatore nella sua scelta facendogli scoprire degli eventuali prodotti che in caso contrario non avrebbero trovato. Tali sistemi, tramite l'acquisizione delle preferenze dell'utente e/o decisioni passate, sono in grado di comprendere le preferenze di tale utente e di condurlo in una scelta più affine ai propri interessi. Ultimo ma non per ultimo motivo di successo dei Sistemi di raccomandazione è il picco di efficenza che hanno portato per molte aziende: automatizzando il processo di suggerimento hanno potuto risparmiare ingenti quantità di denaro.
I sistemi di raccomandazione, come mostrato in **Figura 2.1.**, possono essere suddivisi e categorizzati in 6 diverse macro aree a seconda della metodologia e algoritmo utilizzato.

2.1.1 Collaborative Recommendation Systems

I sistemi di raccomandazione collaborativi sono noti per usare la tecnica di **collaborative filtering (CF)** o filtraggio collaborativo [4].

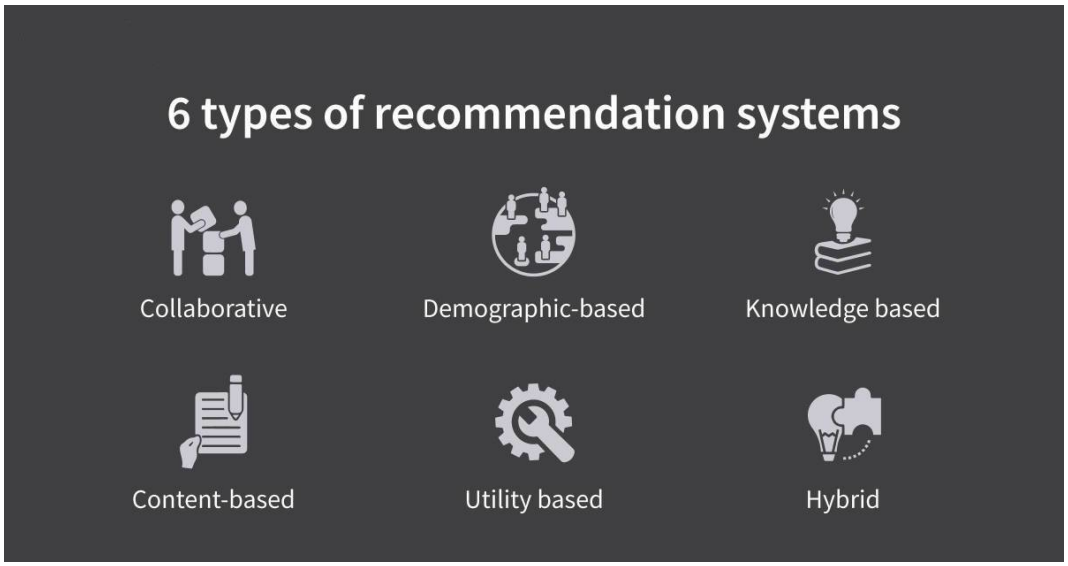

Figura 2.1. Le 6 categorie principali di sistemi di raccomandazione [3]

Tale processo è in grado di filtrare e/o valutare prodotti servendosi dell'aiuto di altre persone che hanno già espresso la loro preferenza su quello stesso prodotto, processo che è sempre esistito nella natura umana ovvero condividere le opinioni con gli altri. Per comprendere a pieno il metodo bisogna introdurre della terminologia che viene condivisa con altre categorie di sistemi di raccomandazione. Il dominio nel quale opera il filtraggio collaborativo è fatto di *utenti* che esprimono preferenze per vari *elementi*. Quando un utente esprime una preferenza per un elemento genera un *punteggio* che viene espresso con la tripla $(Utente, Elemento, Punteggio)$ e può essere rappresentato in varie scale (ad es. da 0 a 5 oppure mi piace/non mi piace) e il set di tutte le triple costituisce la *matrice dei punteggi*, un esempio può essere osservato nella **tabella 2.1.**

	Avatar	The Avengers	Passengers
Utente A	4	5	?
Utente B	1	3	3
Utente C	?	?	2

Tabella 2.1. Esempio di Matrice di punteggio basata su una scala da 0 a 5.

Per valutare l'efficienza di tale sistema di raccomandazione bisogna considerare due compiti importanti:

1. **Prevedere**: dato un utente e un elemento, quale sarà la preferenza e quindi il punteggio per quell'elemento? Nel caso della matrice si riconduce al problema dei valori mancanti.

2. **Suggerire**: dato un utente riuscire a produrre una lista dei migliori n elementi che soddisfano meglio i bisogni di tale utente. Tuttavia non è detto che tale lista contenga gli n elementi con preferenza prevista maggiore in quanto potrebbe non essere l'unico discriminante nel sistema di raccomandazione.

Il primo metodo di filtraggio collaborativo è stato l'algoritmo **User-User Collaborative Filtering**. L'obiettivo principale è trovare altri utenti i quali andamenti dei punteggi passati sono *simili* a quelli dell'utente attuale e usare tali andamenti per prevedere quali saranno gli elementi di suo gradimento. In definitiva, oltre la matrice dei punteggi tale algoritmo necessita di una **funzione di similitudine** che dia in output una misura di quanto due utenti siano simili e anche un metodo che sfrutti tale similitiduine e i punteggi per costruire le previsioni.
Un esempio di funzione di similitudine è la **correlazione di Pearson** che sfrutta la correlazione statistica tra due utenti.

2.1.2 Content-based Recommendation Systems

L'obiettivo dei sistemi di raccomandazione con un approccio basato sul contenuto è suggerire all'utente elementi simili che gli sono già piaciuti o ha già votato positivamente in passato [5]. Tali sistemi di raccomandazione sfruttando gli attributi degli elementi già valutati positivamente dall'utente sono in grado di costruire un profilo strutturato degli interessi dell'utente che viene usato per consigliare nuovi elementi. Il meccanismo principale è il match tra gli attributi del profilo appena costruito e quelli dell'elemento in questione. Il risultato viene espresso tramite un giudizio che rappresenta quanto l'utente è interessato riguardo tale elemento.
É possibile osservare in **Figura 2.2.** la struttura base di un sistema di raccomandazione con approccio basato sul contenuto.

L'algoritmo applicato in questi sistemi è detto **Information Filtering (IF)** o filtraggio delle informazioni, il cui processo di suggerimento è suddiviso in tre fasi e in ognuna è coinvolto un componente diverso. Il **content analyzer** o analizzatore di contenuti che, grazie a tecniche di feature extraction, si occupa di fare un pre-processing dei dati e rappresentare il contenuto

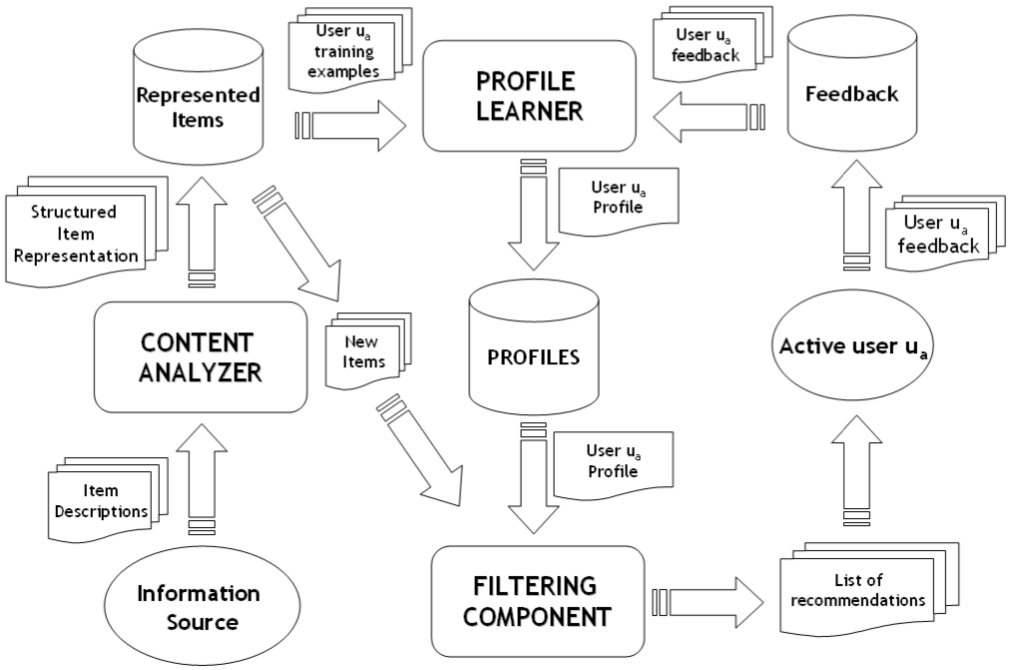

Figura 2.2. Architettura di un Content-Based recommendation system

degli elementi in un formato adatto per le fasi successive del sistema. Il **profile learner** è invece un modulo che registra tutti i dati relativi alle preferenze dell'utente e cerca in qualche modo di generalizzarle per costruire il profilo dell'utente attraverso tecniche di machine learning che sono in grado di generare un modello degli interessi dell'utente avendo come input le preferenze passate già espresse su altri elementi. Infine il **filtering component** o componente di filtraggio, sfruttando il modello dell'utente appena creato, è in grado di suggerire all'utente eventuali elementi affini facendo un confronto tra lo stesso profilo e quello degli elementi suggeriti. L'output di quest'ultimo componente è un giudizio sull'elemento il quale può variare da *mi piace / non mi piace* ad un *punteggio*.
I pregi di tali sistemi risiedono in tre punti fondamentali: è **indipendente dagli utenti** poichè non si basa su altri utenti ma solamente sulle preferenze espresse dall'utente attivo; è **trasparente** perchè è possibile spiegare con facilità il funzionamento semplicemente enumerando le descrizioni che hanno portato un elemento ad essere suggerito piuttosto che un altro; gestisce perfettamente i **nuovi elementi** ed è in grado di suggerirli nonostante non siano stati ancora valutati.

Uno dei difetti invece è la gestione di un **nuovo utente**: è necessario avere un campione significativo di elementi valutati e poichè tale utente potrebbe aver valutato pochi elementi il sistema fa fatica a comprendere le sue preferenze portando ad suggerimenti sub-ottimali.

2.1.3 Demographic based Recommendation Systems

I sistemi basati su un approccio demografico sfruttano gli attributi di un utente e suggeriscono elementi sulla base della **classe demografica**, l'algoritmo utilizzato in questa tipologia di Reccomendation systems è il **demographic filtering (DF)** o filtraggio demografico [6]. L'algoritmo crea i profili degli utenti inquadrandoli in situazioni tipiche astraendo le caratteristiche di una classe di utenti identificati da informazioni demografiche. Il filtraggio demografico genera cluster di utenti accoumnati da simili caratteristiche demografiche e monitora l'andamento delle preferenze e/o degli acquisti cumulativi. Il processo opera come segue: gli utenti vengono divisi nelle loro classi demografiche partendo dalle loro caratteristiche personali, classi che fungeranno da input per l'algoritmo.Le loro caratteristiche vengono acquisite tramite sondaggi o ricavate dalla cronologia di acquisti / preferenze. Lo scopo è riuscire a individuare le classi di utenti ai quali piace lo stesso elemento. É possibile visualizzare in **Figura 2.3.** uno schema generale del flusso esecutivo di questa tipologia di sistemi di raccomandazione [9].

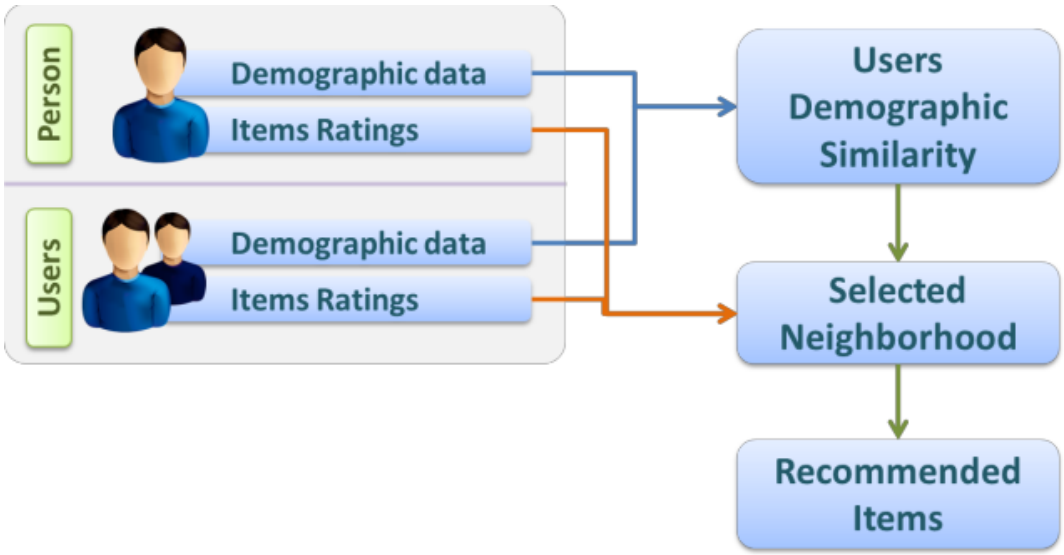

Figura 2.3. Schema generale di un demographic based recommendation system

Definiamo C la classe demografica, c l'utente appartenente alla classe C ed e l'elemento valutato positivamente da tutti gli utenti appartenenti a tale classe. Se l'utente c appartenente alla classe C non ha ancora espresso una preferenza sull'elemento e, l'algoritmo è in grado di suggerire tale elemento per l'utente c. Come nel filtraggio collaborativo, questa categoria di sistemi e più precisamente il filtraggio demografico riesce a creare correlazioni tra utenti ma tramite dati difformi.
Invece a differenza dei content-based, non ci sono problemi nella gestione di un nuovo utente perchè, dopo aver inquadrato tale utente in una classe demografica specifica, i suggerimenti vengono generati a partire dagli acquisti cumulativi di utenti appartenenti della stessa classe, inoltre grande punto di forza è la non obbligatorietà di avere i dati dei punteggi passati degli utenti.

Nonostante sia una tecnica più che valida uno dei suoi grandi punti a sfavore è il giudizio poco specifico e la classificazione troppo generica che portano ad una mancanza di individualità. Inoltre, basandosi sui dati di una classe demografica, tali dati potrebbero essere errati o incompleti portando ad un giudizio non completamente affidabile.

2.1.4 Utility based Recommendation Systems

I sistemi di raccomandazione con approccio basato sull'utilità generano suggerimenti che si basano sul calcolo dell'**utilità** di oggi elemento per l'utente [7]. Le caratteristiche degli elementi sono i dati di input sui quali viene ricavata una **funzione di utilità** che descrive le preferenze degli utenti e successivamente viene applicata per ottenere una classifica di elementi da suggerire per l'utente attivo. Il nucleo computazionale della maggiorparte di questi sistemi risiede nella **MAUT**: Multi-attribute utility theory ovvero una funzione di utilità multi-attributo [8]. Si tratta di una tecnica nella teoria decisionale quando l'agente che deve fare la scelta si trova davanti a poche alternative. Nonostante l'applicazione di MAUT possa variare, i passi principali di questa procedura sono i seguenti:

1. **Definire gli attributi rilevanti e le alternative:** in questo step gli attributi devono essere definiti e scelti poichè le alternative verranno valutate su questi ultimi. É una buona pratica limitarne il numero al fine di evitare fatica e confusione. Il focus di questo step è la ricerca di tali attributi tramite interviste ad-hoc con esperti o reti decisionali. Dopo averli individuati, la scelta viene effettuata tramite una scala naturale quantitativa degli attributi legati all'elemento che si sta analizzando.

2. **Valutare ogni alternativa separatamente per ogni attributo:** dopo aver definito gli attributi rilevanti, le alternative devono essere valutate per ciascuno di questi attributi. Il primo approccio che può essere usato è una **valutazione diretta** dove vengono considerati tre input: due sono considerati come valori limite ed uno invece come giudizio numerico. All'input "negativo" viene assegnato il valore 0, a quello "positivo" il valore 100. L'input rimanente viene giudicato in base ai due già stabiliti. Il secondo approccio invece è la tecnica della differenza delle sequenze standard dove vengono identificati una serie di input posizionati a ugual distanza spaziale. Con la ricerca dicotomica vengono identificati l'input migliore e peggiore e di conseguenza viene selezionato l'input equidistante tra quelli trovati.

3. **Assegnare un peso relativo ad ogni attributo:** viene valutata l'importanza di ogni attributo e successivamente calcolati i pesi. Esistono vari metodi come la classificazione, valutazione diretta, e la tecnica dei pesi oscillanti. Nella classificazione vengono valutati tutti gli attributi in ordine di importanza. Per esempio si potrebbero dividere 100 punti per tutti gli attributi in modo tale che i punti rappresentino l'importanza che acquisice un certo attributo.

4. **Aggregare i pesi degli attributi per ottenere una valutazione complessive :** in questo step vengono aggregati i pesi tramite vari modelli, il più utilizzato tuttavia è la cosiddetta funzione *weighted linear additive preference*. La valutazione finale di un'alternativa viene calcolata tramite la moltiplicazione del peso per il valore dell'attributo per ogni attributo e poi vengono sommati questi valori su tutti gli attributi. L'alternativa con il valore più alto di questa funzione sarà verosimilmente quella che verrà scelta. Il punto di forza della funzione in questione è l'elevata capacità di gestire i conflitti tra i valori poichè viene calcolato il trade-off a seconda della relativa importanza data anche dal peso già calcolato.

5. **Produrre il suggerimento:** in questa fase vengono eseguite delle analisi volte a decidere la stabilità dei risultati e viene finalmente valutata l'importanza dei valori e dei pesi delle alternative disponibili. Sia la votazione diretta che la ricerca dicotomica sono metodi che vengono utilizzati per produrre diversi valori e pesi. Esiste un ulteriore metodo per avere più informazioni circa la stabilità dei risultati: cambiare la funzione che

determina i pesi come per esempio la *equal weight function* che associa ad ogni attributo lo stesso peso. Infine il modello è in grado di produrre in output il suggerimento sulla base di tutte le nozioni sopracitate.

2.1.5 Knowledge based Recommendation Systems

I sistemi di raccomandazione basati sulla conoscenza o anche detti **knowledge based** cercano di suggerire gli elementi inferendo sulle preferenze e necessità di ogni utente [10]. Questi sistemi sono piuttosto robusti e non soffrono del problema della *cold start* o "partenza fredda" nei casi quando la disponibilità dei dati è molto limitata oppure non si ha una visione collettiva di tutte le combinazioni utente-elemento. I knowledge based recommendation systems si basano sull'estrazione esplicita dei requisiti da parte dell'utente. Poichè in molte situazioni l'utente però potrebbe non essere completamente al corrente di tutte le variabili di un dato elemento / prodotto, tali sistemi sono dotati di un *feedback interattivo* tramite il quale l'utente può esplorare la complessità dei prodotti e imparare a valutare la convenienza di uno rispetto ad un altro. Il processo di estrazione e di esplorazione è facilitato dalla conoscenza del dominio del prodotto ed è così importante al fine di produrre un suggerimento valido che tali sistemi hanno preso il nome knowledge based.
I sistemi di raccomandazione basati sulla conoscenza sono più performanti su prodotti che non vengono comprati con una certa frequenza come per esempio le automobili e in questi casi in media l'utente è più selettivo e esplicito circa le sue richieste. In **Figura 2.4.** è possibile osservare il metodo di questa tipologia di recommendation systems.

In generale, le categorie di prodotti nei quali si possono utilizzare tecniche basate sulla conoscenza differiscono da quelle dei sistemi di raccomandazione enunciati nei paragrafi precedenti, pertanto di seguito vengono descritte le situazioni dove i knowledge based sono più performanti:

- Gli utenti vogliono esplicitare i loro requisiti e quindi l'interattività del sistema è di vitale importanza

- A causa della complessità della categoria di elementi è difficile recuperare i punteggi di un elemento specifico sopratutto in termini di tipologie e opzioni disponibili

- In alcune categorie come i computer o le automobili dove i punteggi sono inclini ad una grande variazione nel corso del tempo

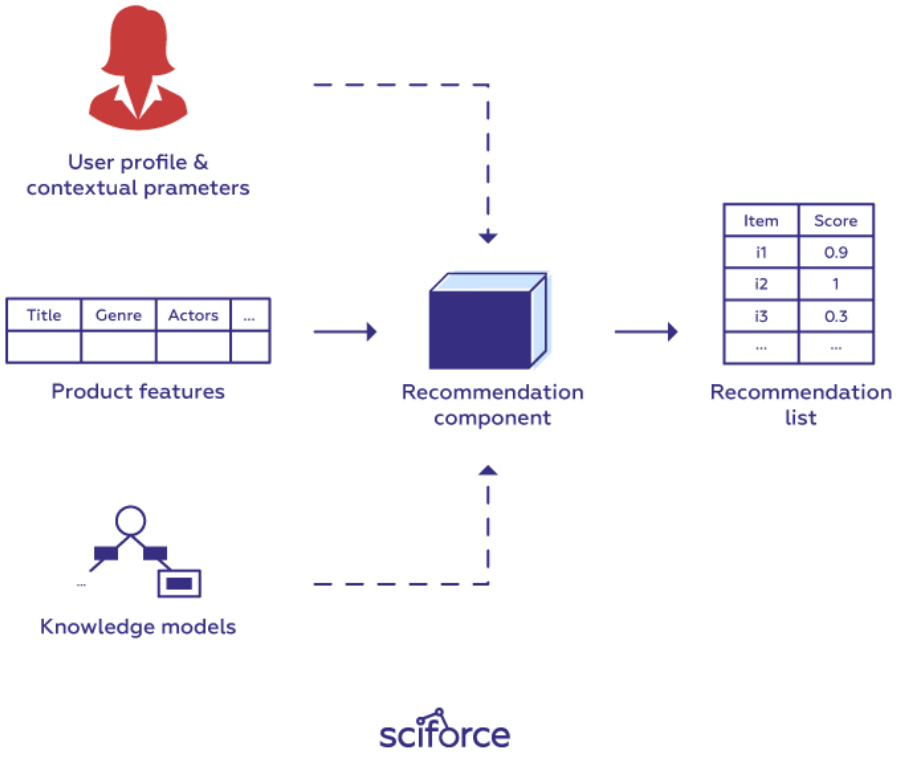

Figura 2.4. Schema di un knowledge based recommendation system

Un'altra importante caratteristica che distingue i knowledge based rispetto agli altri è l'elevata personalizzazione resa possibile grazie alla conoscenza della categoria di prodotti che si traduce in vincoli e metriche di similarità. Per questo motivo è possibile dividerli in due categorie a seconda della metodologia interattiva e della conoscenza usata per facilitarla:

1. **Constraint-based:** cercano di fare da collegamento tra le richieste dell'utente (vincoli) e le caratteristiche dell'elemento. Quindi l'utente può specificare dei vincoli sugli attributi dell'elemento e inoltre vengono usate delle regole specifiche per cercare di fare match tra gli attributi dell'utente e quelli dell'elemento. Queste regole non sono altro che la conoscenza specifica del dominio usato dal sistema. Spesso capita che gli utenti non siano in grado di specificare le loro esigenze in base agli attributi che descrivono i prodotti e per far fronte a questo problema viene introdotto un nuovo set di regole che mette in relazione gli attributi degli utenti

con quelli degli elementi.

2. **Case-based:** vengono utilizzate delle metriche di similarità per collezionare esempi che sono simili agli elementi target specificati, detti anche *cases*. Le metriche di similarità sono spesso legate alla categoria di elementi presi in considerazione e quindi sono proprio le metriche che costituiscono la base di conoscenza di questi sistemi. I suggerimenti prodotti in output sono spesso trasformati nei nuovi target ma con delle modifiche effettuate interattivamente dall'utente.

2.1.6 Hybrid Recommendation Systems

I sistemi di raccomandazione **ibridi** descrivono tutti quei sistemi che mettono insieme più di una tecnica di raccomandazione sopracitate per produrre un suggerimento in output [11]. Nonostante si possano combinare due sistemi di racomandazione basati sulla stessa tecnica (ad es. collaborative-filtering) il risultato potrebbe essere ridondante, motivo per cui ci si concentra sui sistemi che combinano informazioni da diverse fonti. Tra i tanti tipi se ne possono identificare quattro principali:

Weighted

È il più semplice dei sistemi ibridi dove ogni componente del sistema da in output un elemento con un punteggio e successivamente, per ogni componente, vengoo combinati i punteggi tramite formula lineare. Un esempio di algoritmo weighted può essere immaginato come in **Figura 2.5.** Un algoritmo pesato opera in varie fasi, nella fase di *training* ogni componente del sistema ibrido elabora i dati di training. Successivamente quando una predizione viene generata per un utente, il sistema propone dei candidati, fase neccessaria per capire quali elementi verranno considerati. Nella fase finale di valutazione, i candidati vengono valutati congiuntamente. Come ultima fase i candidati vengono ordinati in ordine di punteggio crescente e i primi elementi vengono suggeriti all'utente.

Switching

Per questa tipologia di sistemi di raccomandazione ibridi viene scelto un singolo sistema di raccomandazione tra tutti i suoi componenti basato sulla situazione specifica. Potrebbe essere selezionato un sistema diverso per ogni

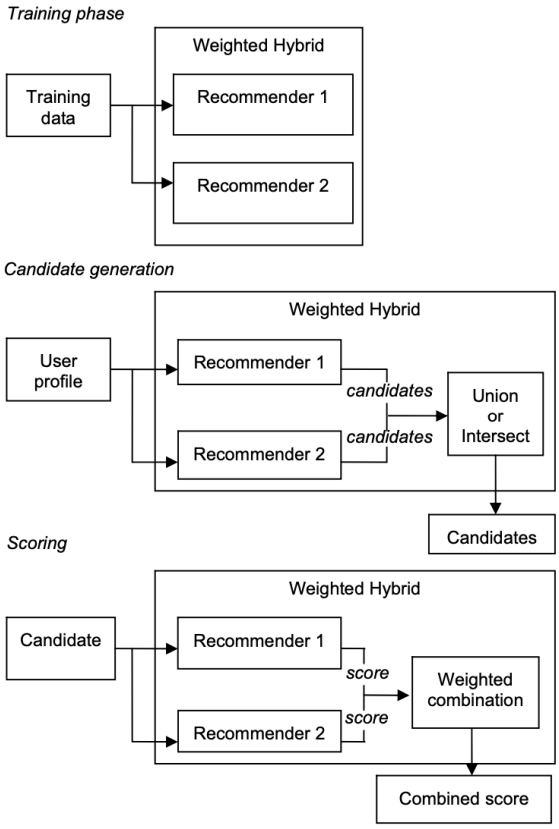

Figura 2.5. Sistema di raccomandazione ibrido weighted

utente diverso. Questa tipologia tiene conto dell'eventuale mancanza di performance consistente per tutte le tipologie di utenti. La scelta del sistema si basa su dei criteri di cambio. Una volta che la scelta è stata fatta, i componenti non scelti non giocano nessun ruolo nel modello decisionale. Schema e architettura sono visibili in **Figura 2.6.**

Feature Combination

L'idea di questi sistemi sta nell'inserire caratteristiche da una sorgente (ad es. collaborative filtering) come input per l'algoritmo programmato per elaborare i dati da una sorgente diversa (ad es. demographic-based). Viene aggiunto un componente virtuale chiamato "sistema di racomandazione contribuente": tutti gli attributi che dovrebbero essere processati da questo sistema vengono invece reindirizzati come input del sistema principale, motivo per il quale si ha un *mix* di caratteristiche. É un modo per potenziare la capacità di un

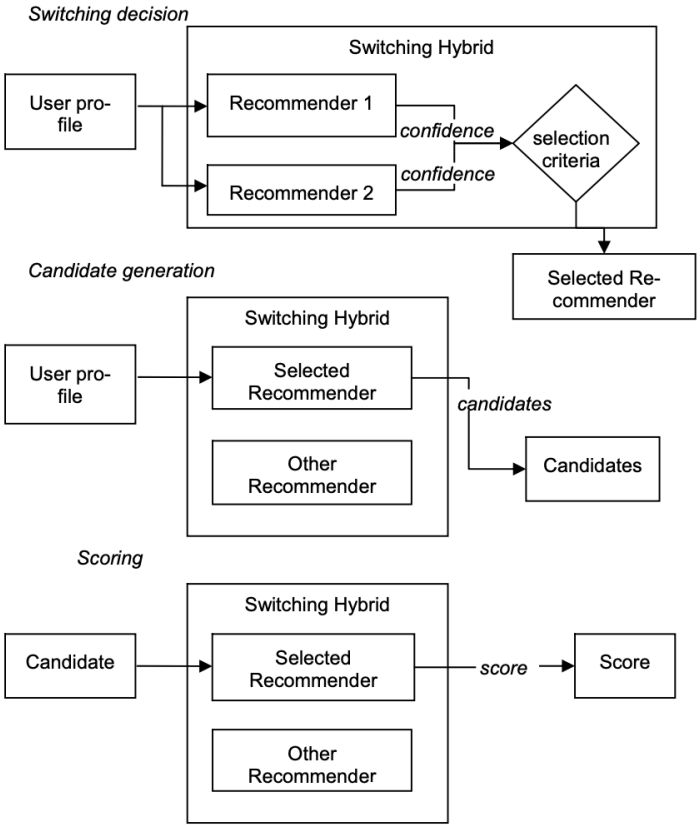

Figura 2.6. Sistema di raccomandazione ibrido switching

sistema già a tutto tondo. É possibile osservare in **Figura 2.7.** una visione complessiva di tali sistemi.

Feature Augmentation

Sistemi molto simili ai feature combination, differiscono nel modo in cui vengono scelte le caratteristiche: sono generate per ogni nuovo elemento usando la logica di raccomandazione del "sistema di racomandazione contribuente". In ogni fase, tale sistema intercetta i dati che dovrebbero essere spediti al sistema principale e li "aumenta" generandone altri, differenza principale con i feature combination dove vengono solo reindirizzati. Il processo di aumento dei dati può essere eseguito spesso offline modificando a proprio piacimento i dati di input dell'algoritmo. Sono sistemi particolarmente performanti quando il sistema di raccomandazione primario è programmato bene e si vuole avere una fonte aggiuntiva di dati.

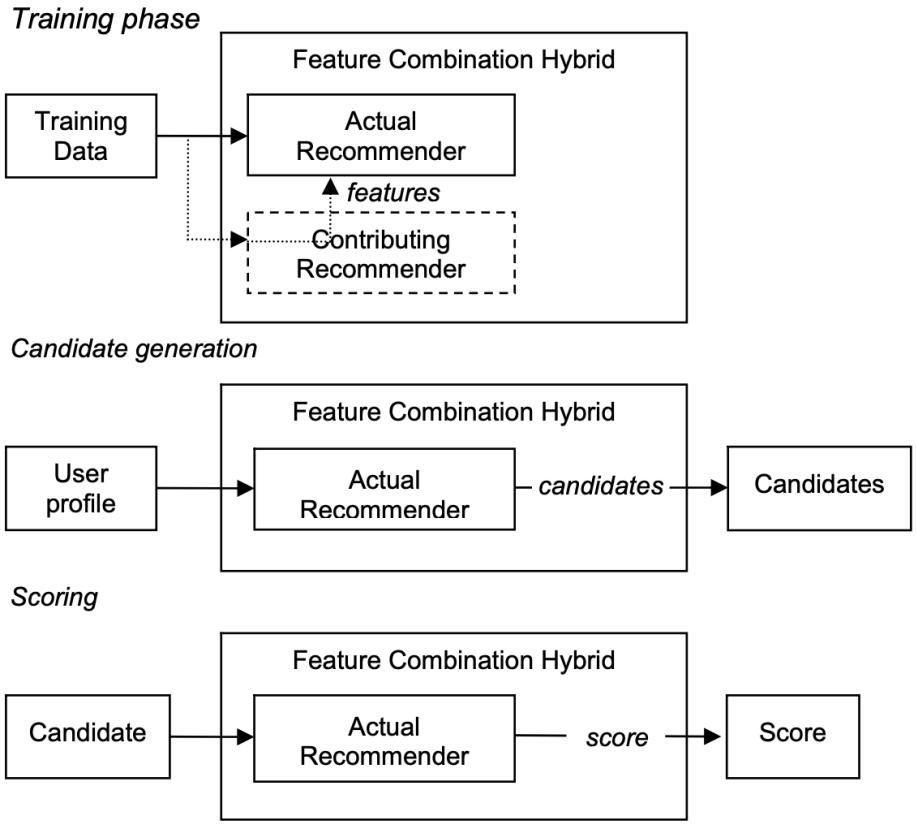

Figura 2.7. Sistema di raccomandazione ibrido con feature combination

2.2 Piattaforme mobile concorrenti

In questa sezione verranno brevemente discusse e esplorate alcune piattaforme mobile concorrenti alternative ad oggi con scopi e obiettivi simili a questa tesi.

2.2.1 Petfinder Mobile

Petfinder è un database online di animali domestici tra cui cani, gatti, conigli e pesci che hanno bisogno di una casa. in collaborazione con oltre 11.000 rifugi e associazioni in tutti gli Stati Uniti. Come missione hanno interesse ad aumentare la consapevolezza del numero di animali lasciati nei canili e di aumentare l'efficacia dei programmi di adozione in tutto il Nord America. Iniziato come progetto web, di recente hanno rilasciato la loro applicazione negli store principali con le seguenti caratteristiche:

- Filtrare a partire dalle caratteristiche dell'animale domestico e delle proprie abitudini e routine
- Visualizzare foto e video di cani e gatti
- Condividere i profili di tutti gli animali domestici
- Vedere i dettagli di come il tuo profilo possa formare un "match" con quello di un animale domestico
- Possibilità di contattare facilmente rifugi e associazioni per proseguire con il processo di adozione

2.2.2 We Rescue - Adopt a Pet

We Rescue è un'applicazione mobile volta a fare trovare agli utenti situati negli Stati Uniti e in Canada il loro nuovo animale domestico. É possibile cercare cani, gatti, cavalli, conigli, rettili di qualsiasi razza. Gli animali domestici presenti in molti rifugi e associazioni di volontariato sono mostrati in base alla distanza. Di seguito i punti distintivi dell'app:

- Visualizzazione completa dei dettagli per ogni animale domestico
- Condivisione nei social media tra cui Pinterest, Facebook, Twitter degli animali presenti nel database
- Possibilità di aggiungere ai preferiti i rifugi o associazioni di volontariato
- Ricerca avanzata per razza, età, taglia, sesso e caratteristiche che si adattano al tuo stile di vita
- Ponte diretto tra l'utente e l'associazione
- Sezione dedicata sull'adozione, su come comportarsi e quali sono i prossimi step
- Partner ufficiale di "Clear the Shelters"

2.2.3 ResQwalk

Nonostante non sia propriamente un'applicazione per adottare direttamente animali domestici e quindi un vero concorrente, è degna di nota per la sua idea originale. ResQwalk ti permette di raccogliere fondi per rifugi e associazioni semplicemente facendo una passeggiata. Ogni settimana viene annunciata la somma che verrà data in beneficenza. Per far guadagnare soldi al tuo rifugio preferito è necessario l'uso del GPS quando viene fatta la passeggiata. Alla fine della settimana ResQwalk dona il quantitativo corrispondente in proporzione alla distanza totale percorsa durante tutta la settimana. In questo modo è possibile sia salvare e migliorare le vite di tutti gli animali domestici senza una casa.

Capitolo 3

Tecnologie

Il punto focale di questo capitolo è approfondire e illustrare al lettore le tecnologie utilizzate per lo sviluppo della piattaforma, oggetto di questa tesi magistrale. Il capitolo è stato diviso in due per riflettere la divisione della piattaforma stessa: nella prima parte si parlerà del framework utilizzato per sviluppare l'applicazione mobile, il cosiddetto frontend. Successivamente verranno illustrate al lettore alcune alternative che si sarebbero potute utilizzare. Nella seconda parte invece si andrà nel dettaglio del backend e del linguaggio di programmazione utilizzato per realizzare il lato server sul quale l'app si appoggia. Come per la prima parte, anche per la seconda verranno infine mostrati pro e contro di altri framework e database non utilizzati per lo sviluppo.

3.1 Applicazione mobile

L'applicazione mobile, oggetto di questa Tesi Magistrale, è stata realizzata interamente in **Flutter**. É una combinazione di due elementi: è un **framework** UI ovvero una raccolta di pacchetti di codice e funzioni di utilità per sviluppare applicazioni **multipiattaforma**, principale punto di forza di Flutter. Da un solo codice sorgente è possibile generare applicazioni per un'ampia varietà di piattaforme di destinazione come mostrato in **Figura 3.1.** Il secondo elemento che completa la definizione è un **insieme di strumenti** (ad es. un'interfaccia a riga di comando) che aiutano a sviluppare, testare e scrivere del codice che sia in grado di funzionare su diverse piattaforme di destinazione poichè in realtà il codice sorgente non è in grado di essere eseguito immediatamente su tutte le piattaforme ma deve essere prima tradotto in codice macchina che funzioni su queste diverse piattaforme. La

traduzione da codice sorgente a codice machina viene fatta da Flutter e più specificatamente dall'insieme di strumenti sopracitati.

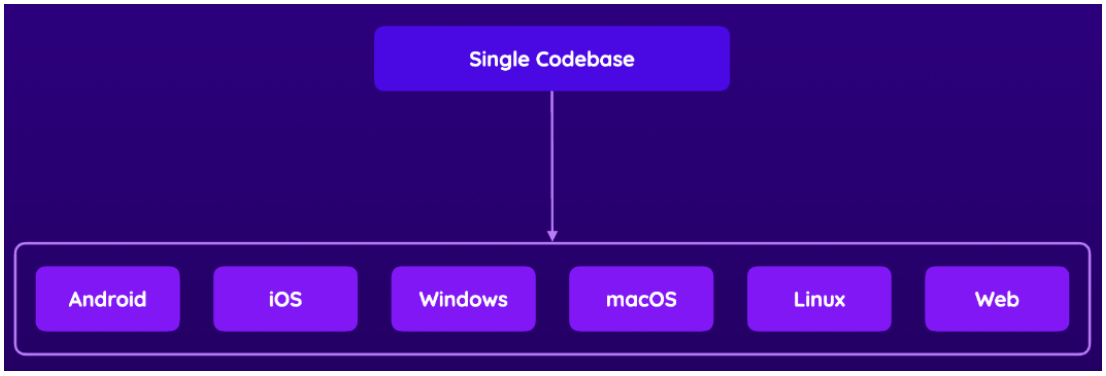

Figura 3.1. Piattaforme nelle quali può essere tradotto il codice sorgente di Flutter

3.1.1 Flutter setup

Per iniziare a programmare in Flutter è necessario scaricare il **Flutter SDK** ovvero il kit di sviluppo software che contiene Flutter stesso. É necessario anche installare **Git** ovvero un software di version control che generalmente è totalmente indipendente dal linguaggio di programmazione ma in realtà l'insieme di strumenti di Flutter usano questo software Git internamente, motivo per il quale è necessario installarlo.
Per poter costruire l'applicazione su piattaforme diverse e per poterne visualizzare l'anteprima mentre ci si lavora, è necessario installare anche alcuni strumenti specifici per la piattaforma, i cosiddetti **platform tools**. Per esempio sarà necessario scaricare **Android Studio** per sviluppare e testare applicazioni su Android, **XCode** invece per le applicazioni che verranno usate nei dispositivi aventi iOS come sistema operativo. L'ultimo passo per poter cominciare a sviluppare in flutter è impostare e installare alcuni dispositivi virtuali ed emulatori sul proprio sistema. Ad esempio un *emulatore Android* per visualizzare l'anteprima del progetto Flutter come un'applicazione Android che viene eseguita localmente sul dispositivo virtuale della propria macchina.

3.1.2 Dart

Tecnicamente, Flutter non è un linguaggio di programmazione ma un **framework** (un insieme di pacchetti e funzioni di utilità che si possono usare nel codice) per la costruzione di interfacce utente tramite **dart** ovvero il linguaggio di programmazione vero e proprio. É stato sviluppato da Google che, nonostante Flutter sia il suo caso d'uso principale, può essere usato anche al di fuori dello sviluppo di applicazioni mobile o multipiattaforma. É un linguaggio di programmazione orientato agli oggetti, basato sulle classi e garbage-collected ovvero con un sistema automatico di gestione della memoria.
Dart rispetta la sicurezza rispetto ai tipi (type safe): tramite un controllo sui tipi statici è in grado di garantire che il valore di una variabile sia sempre uguale al suo tipo statico, questo meccanismo spesso è chiamato **sound typing**. Un altro punto di forza di Dart è la **null safety**: i valori delle variabili non possono essere nulli a meno che non sia il programmatore stesso a decidere ciò. Questa caratteristica previene dalle eccezioni a runtime grazie ad una analisi statica del codice. La tecnologia del compilatore di Dart permette di eseguire il codice in due modalità come anche graficamente mostrato in **Figura 3.2.** :

- **Piattaforma nativa:** per tutte quelle app sviluppate per sistemi mobile e desktop, Dart include la sua macchina virtuale **Dart VM** con traduzione dinamica effettuata durante l'esecuzione del programma piuttosto che precedentemente e un compilatore **AOT** in grado di compilare il codice prima dell'esecuzione del programma, di solito nella fase di build.

- **Piattaforma web:** per tutte le app sviluppate per il web, Dart compila sia per scopi di sviluppo o di produzione. Il suo compilatore web traduce Dart direttamente nel linguaggio JavaScript.

Indipendentemente da quale piattaforma viene usata o come viene compilato il codice, eseguirlo richiede il **runtime** di Dart che gestisce i seguenti processi critici:

1. **Gestione della memoria:** dart usa un modello di gestione della memoria dove quella non utilizzata viene riacquisita tramite il suo garbage collector.

2. **Imporre il sistema di tipizzazione:** anche se molti dei controlli dei tipi sono statici (effettuati al tempo di compilazione) alcuni tipi invece sono *dinamici* e vengono eseguiti al runtime.

3. **Gestione degli "isolates":** sono, come dei processi dove però ognuno di essi ha la sua memoria e un thread principale. Il runtime di dart li controlla tutti, sia il main che tutti quelli che l'app crea.

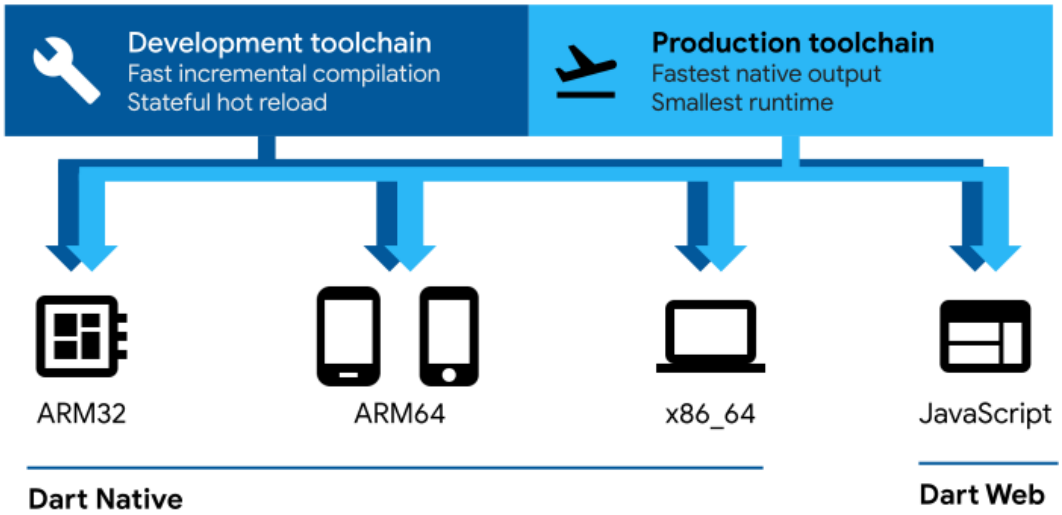

Figura 3.2. Piattaforme Dart

3.1.3 Architettura di Flutter

Flutter è stato concepito come un sistema estendibile a più livelli. É costitutito da una serie di librerie indipendenti le quali si basano sul livello sottostante. Nessuno dei livelli ha accessi privilegiati al livello inferiore e ogni parte del sistema è opzionale e sostituibile. In Figura 3.3. vengono descritti tutti i livelli e le funzionalità.

Le applicazioni in Flutter, dal punto di vista del sistema operativo sottostante, vengono viste come un normalissimo pacchetto come potrebbero esserlo le applicazioni native. Un embedder specifico per piattaforma si coordina con il sistema operativo per gestire servizi come accessibilità e input. Di solito è scritto nel linguaggio di programmazione più appropriato per ogni piattaforma che sia Android (Java e C++) o iOS (Objective-C). L'embedder

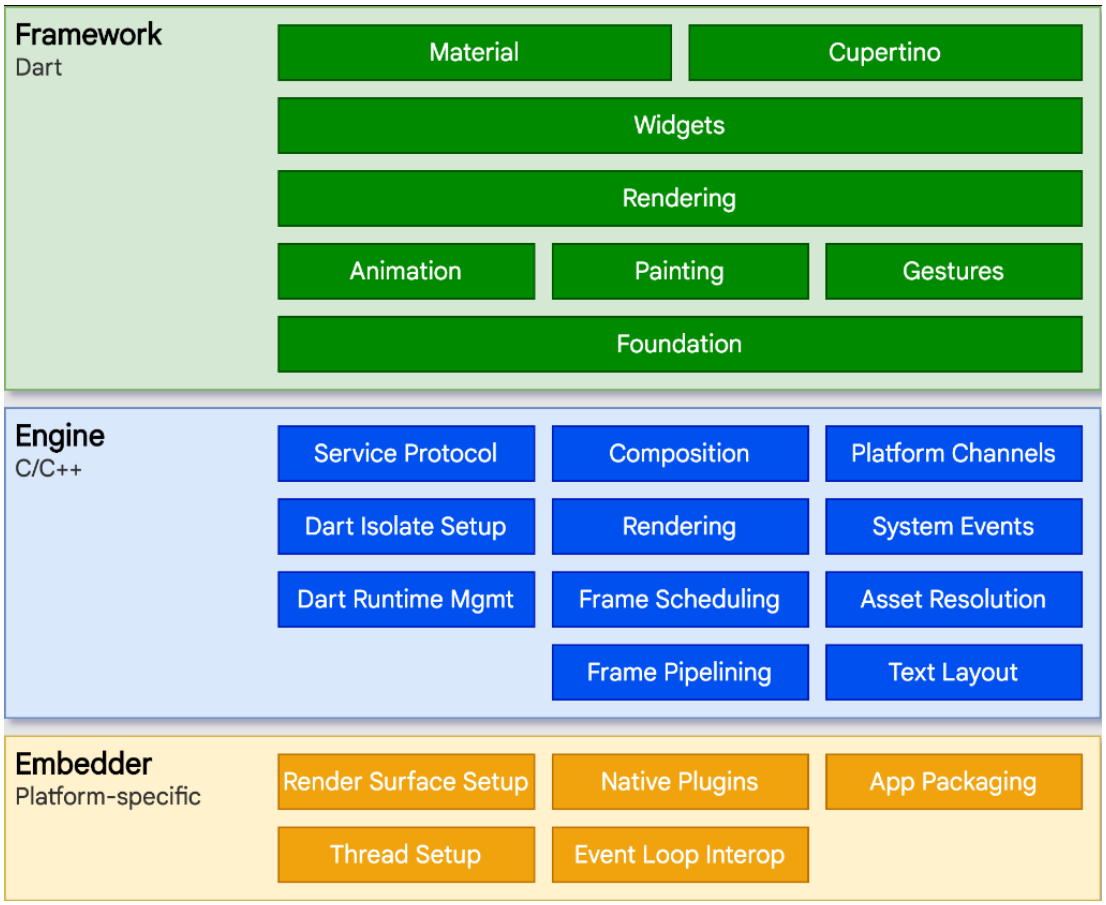

Figura 3.3. Architettura di Flutter

permette che il codice in flutter sia integrato in una applicazione esistente come un modulo.
Il cuore pulsante dell'architettura è il motore di flutter o **Flutter engine**. Scritto nella sua quasi interezza in C++ contiene tutte le primitive necessarie per supportare le applicazioni. Fornisce le implementazioni di basso livello di grafiche, layout di testi, accessibilità, ecc. Il motore è reso disponibile nella libreria `dart:ui` che impacchetta in Dart il codice sottostante scritto in C++.
Il livello con cui i programmatori si interfacciano con Flutter è sicuramente il **Flutter Framework** scritto in Dart e comprende:

- **Foundation** o classi base e servizi come animazioni che astraggono le classi base sottostanti

- Il **rendering layer** che permette di creare un albero di oggetti che possono essere mostrati a schermo, manipolati dinamicamente tramite un aggiornamento automatico che rispecchia gli eventuali cambiamenti imposti dal programmatore

- Il **widget layer** ovvero un'astrazione degli oggetti appartenenti al livello di rendering. Il modello di programmazione reattiva è introdotto qui.

- Le librerie **Material** e **Cupertino** che offrono strumenti per manipolare le composizioni di oggetti facenti parte il widget layer implementando lo stile Material o simil iOS.

3.1.4 Widgets

I **widget** sono l'unità compositiva di Flutter, gli elementi base che costituiscono l'interfaccia utente di un'applicazione scritta in Flutter e ogni widget è una dichiarazione immutabile di una parte di tale interfaccia utente. Essi formano una gerarchia basata sulla composizione: ogni widget è ospitato da un widget padre ed eredita il suo contesto. Si forma così una catena di widget fino ad arrivare al widget **radice** ovvero il contenitore dell'intera app Flutter (di solito MaterialApp o CupertinoApp in base allo stile che si vuole dare all'applicazione).
Le applicazioni in Flutter aggiornano la loro interfaccia utente a seguito di eventi riferendo al framework di sostituire un widget nella gerarchia con un altro. Successivamente il framework fa un confronto tra i nuovi widgets e i vecchi e aggiorna l'interfaccia. Flutter fornisce una propria implementazione per ogni elemento dell'interfaccia utente di cui si possono elencare i pregi:

- **Estensibilità illimitata:** chi programma non è limitato dai punti di estensione del sistema operativo.

- **Evita il problema del collo di bottiglia:** non è necessario che il framework passi da codice Flutter a codice legato alla piattaforma e viceversa.

- **Disaccoppia il comportamento dell'applicazione dalle dipendenze dei sistemi operativi:** l'applicazione è la stessa su tutte le versioni dei sistemi operativi.

3.1.5 Alternative per lo sviluppo mobile

In un'era in cui lo sviluppo di applicazioni mobile è un business sempre in crescita, dove virtualmente la maggiorparte delle persone ha ormai un telefono e vengono sviluppate app per qualsiasi cosa, i frameworks, linguaggi di programmazione e strumenti per sviluppare si moltiplicano sempre più. Per questo motivo, in questo paragrafo verrà discussa la principale alternativa con la quale vengono spesso fatti confronti con Flutter ovvero **React Native**.
React Native è un framework open source usato per sviluppare applicazioni Android o iOS usando **React**: libreria dichiarativa, efficiente e flessibile scritta in JavaScript per costruire interfacce utente tramite piccoli ed isolati blocchi di codice chiamati *componenti*.

React Native

Con React Native viene sfruttato il linguaggio JavaScript sia per accedere alle API della piattaforma ma anche per descrivere l'aspetto e il comportamento dell'interfaccia utente sfruttando i componenti React ovvero pacchetti di codice riutilizzabile.
Nello sviluppo di un'applicazione Android e iOS, una **vista** è un blocco base dell'interfaccia utente e può essere usata per mostrare sullo schermo testi, immagini e altro. In React Native è possibile invocare le viste native dei linguaggi di sviluppo mobile (Kotlin/Java per Android, Swift/Objective-C per iOS) tramite JavaScript usando i componenti di React poichè tali viste native vengono costruite da React Native al runtime. Questi componenti vengono chiamati quindi **componenti nativi**. I componenti nativi essenziali che React Native offre per cominciare a mettere in piedi la propria app vengono chiamati componenti nucleo o **core components**. In Figura 3.4 è mostrato un diagramma della gerarchia dei componenti in React Native.

Flutter vs. React Native

La differenza principale tra Flutter e React Native è che il codice di quest'ultimo non viene compilato nel linguaggio nativo mobile ma esegue il suo codice in JavaScript. Flutter invece compila il codice in Dart e lo traduce in linguaggio nativo. Mentre le app create con Flutter sono indistinguibili da quelle native, il problema principale invece con React Native è legato al

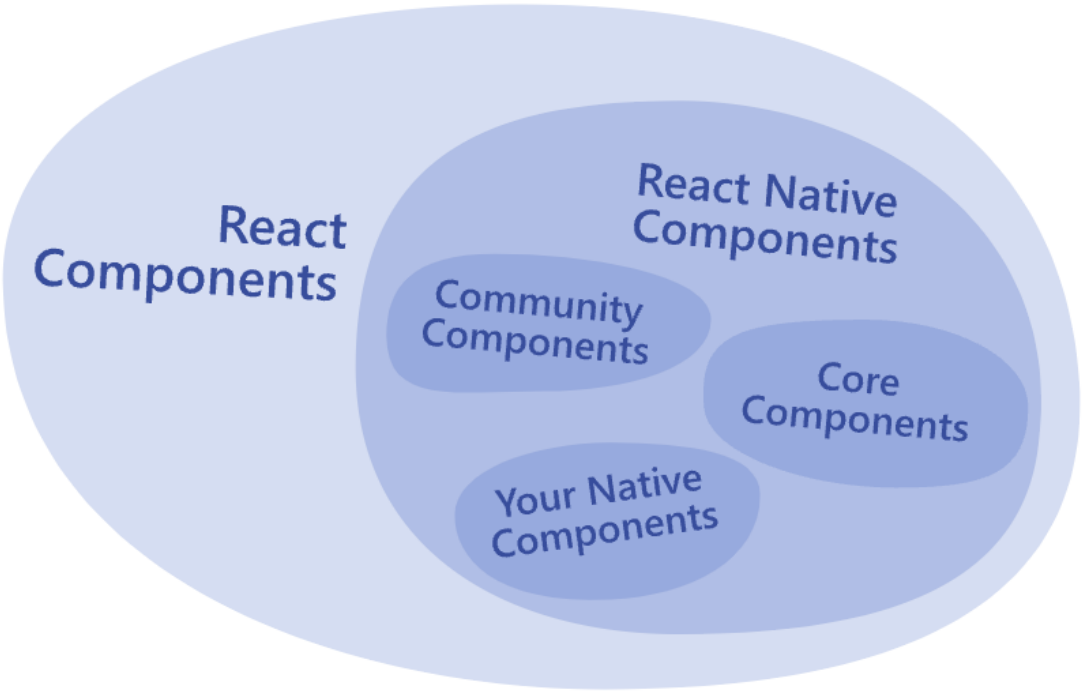

Figura 3.4. Diagramma dei componenti in React Native

suo ambiente di runtime molto pesante. Questo significa che non sarà possibile raggiungere l'efficienza di un'applicazione puramente nativa. Di seguito verranno elencate alcune caratteristiche che differiscono tra i due frameworks:

- **Riuso del codice tra piattaforme mobile:** le applicazioni in Flutter sono sviluppate con codice specifico per la piattaforma quindi non è possibile condividerlo tra applicazioni iOS e Android. Esistono comunque librerie di terze parti che facilitano il riuso di componenti nativi. React Native invece ha le sue API che possono essere usate sviluppando sia per iOS che per Android, è quindi possibile programmare componenti condivisi tra i due sistemi operativi appena citati.

- **Specifiche minime richieste dall'SDK:** React Native può essere usato per creare app per versioni più vecchie di iOS (9+) e dell'SDK di Android (5.0+) con un sottoinsieme limitato di tutte le funzionalità disponibili al runtime. Flutter, come React Native, può anch'esso essere impiegato nella creazione di app per versioni di sistemi operativi più vecchi (iOS 8+, Android Lollipop in su) però nel caso di iOS la chiamata

di alcune API potrebbe causare dei errori al runtime a causa di molte API deprecate da parte della Apple.

- **Debugging:** Flutter ha il suo debugger che, agganciandolo all'applicazione sia su iOS che su Android, fornisce informazioni sullo stato del motore di rendering e offre vari strumenti di gestione memoria e altro. Anche React Native ha il suo debugger che da più informazioni però sullo stato della macchina virtuale di JavaScript.

- **Dimensione dell'app:** mentre Flutter integra un compilatore AOT che comprime solo il codice sorgente necessario per far funzionare correttamente l'applicazione, React Native invece include con se il runtime di JavaScript di circa 300kb compressi.

- **Sviluppo UI:** Flutter ha il suo set di widgets per renderizzare l'interfaccia utente ed è possibile riutilizzare codice iOS o Android già esistente. React Native offre un "ponte" sotto forma di codice JavaScript in grado di riusare estratti di codice iOS o Android, inoltre alcune APIs possono essere sfruttate per creare manualmente il ponte tra i componenti nativi e il codice in JavaScript.

- **Rendimento nativo:** il compilatore AOT di Flutter produce codice ottimizzato sia per iOS che per Android una volta buildato il progetto. Le prestazioni quindi di un'app in Flutter sono al pari di un'app nativa. Anche React Native con la sua macchina virtuale JavaScript è in grado di ottenere la stessa performance per esempio di un'applicazione per iOS senza dover cambiare le impostazioni della build.

3.2 Backend

Il backend, detto anche lato server, è la parte nascosta agli utenti. Nello sviluppo web e mobile si tratta di tutte le logiche necessarie ad un corretto funzionamento dell'applicazione, logiche nascoste e non visibili all'utente finale. Alcune delle sue funzioni sono il collegamento col database, interazioni con le API e il corretto funzionamento del frontend. Il backend di questo progetto di Tesi Magistrale è stato realizzato tramite **Django**. Si tratta di un web framework di alto livello scritto in **Python**, linguaggio di programmazione orientato agli oggetti, in grado di garantire uno sviluppo rapido, sicuro e manutenibile di siti web.

3.2.1 Django Framework

Di seguito verranno elencati i punti di forza principali di Django che permette di sviluppare un software:

- **Scalabile:** l'architettura di Django è del tipo **shared-nothing** ovvero ogni componente è indipendente dagli altri e quindi può essere prontamente sostituito e cambiato se serve. Data questa separazione è evidente la scalbilità dell'archiettura all'aumentare del traffico dati, basta aggiungere un hardware in qualsiasi livello che siano server database, server che utilizzano la cache o server applicativi.

- **Sicuro:** Django aiuta gli sviluppatori fornendo un framwork in regola con un meccanismo automatico di protezione del sito web. Per esempio Django fornisce un modo sicuro per la gestione dell'utente e delle password evitando errori comuni come inserire in punti vulnerabili, nei cookies, le informazioni sulla sessione. Inoltre il framework è robusto rispetto a SQL injection e cross site scripting (problema che persiste nei siti che fanno un uso non sicuro di forms).

- **Portatile:** dal momento che Django è scritto in Python, linguaggio di programmazione in grado di essere eseguito su molte piattaforme, non è legato a nessuna piattaforma server. Inoltre Django è supportato da molti hosting provider che spesso forniscono strumenti specifici per ospitare siti programmati con il framework.

- **Completo:** Django è un framework completo e offre agli sviluppatori tutto il necessario per sviluppare un'applicazione web e non solo. Il tutto funziona con una certa sinergia poichè tutti gli strumenti necessari fanno parte di un prodotto unico e anche grazie alla documentazione sempre aggiornata.

- **Manutenibile:** il codice in Django è scritto in modo tale da usare principi e pattern volti alla manutenibilità e al riuso. Non ci sono rindondanze non necessarie e pertanto il codice è ridotto al minimo indispensabile. Inoltre Django raggruppa insieme funzionalità simili in *applicazioni* che si traduce, ad un livello più basso, raggruppare il codice nei moduli secondo il paradigma **Model View Controller** che verrà trattato nel prossimo paragrafo.

- **Versatile:** Django può essere usato per realizzare qualsiasi sito web e si interfaccia con qualsiasi framework lato client. Internamente può essere anche esteso se neccessario al fine di usare altri componenti.

Infine, Django è un framework semi dogmatico o meglio "somewhat opinionated". Fornisce un set di componenti per gestire la maggiorparte dei compiti durante uno sviluppo web e insieme ad essi presenta uno o due modi per usarli.

MVC

Il Model-View-Controller o **MVC** è un pattern nella progettazione del software usato per implementare interfacce utente, dati e la logica sottostante [12]. Le tre separazioni, descritte in **Figura 3.5.**, consistono in:

1. **Model:** gestisce i dati, definisce cosa debbano contenere e anche la logica. Se lo stato di questi dati cambia, il model notifica la view in modo tale che possa cambiare la visualizzazione e a volte notifica anche il controller se è necessaria una logica diversa per controllare la view aggiornata.

2. **View:** definisce come i dati dell'applicazione debbano essere mostrati a video.

3. **Controller:** contiene la logica che aggiorna il model e/o la view a seguito di un input da parte dell'utente. A volte è necessario aggiornare la vista per mostrare i dati in un formato diverso, in questo caso il controller gestisce direttamente l'aggiornamento senza passare dal modello.

Architettura

Nei siti web tradizionali, un'applicazione aspetta che si verifichi una richiesta HTTP dal browser. Quando questa richiesta viene ricevuta, l'applicazione recupera le informazioni necessarie che ricava dall'URL ed eventuali aggiunte nei dati della richiesta GET o POST. Successivamente, a seconda dei dati da recuperare, l'applicazione potrebbe leggere o scrivere dal database. Infine l'applicazione risponderà al browser creando dinamicamente una pagina HTML in modo tale da mostrare a video i dati recuperati riempiendo i placeholders in un modello HTML.

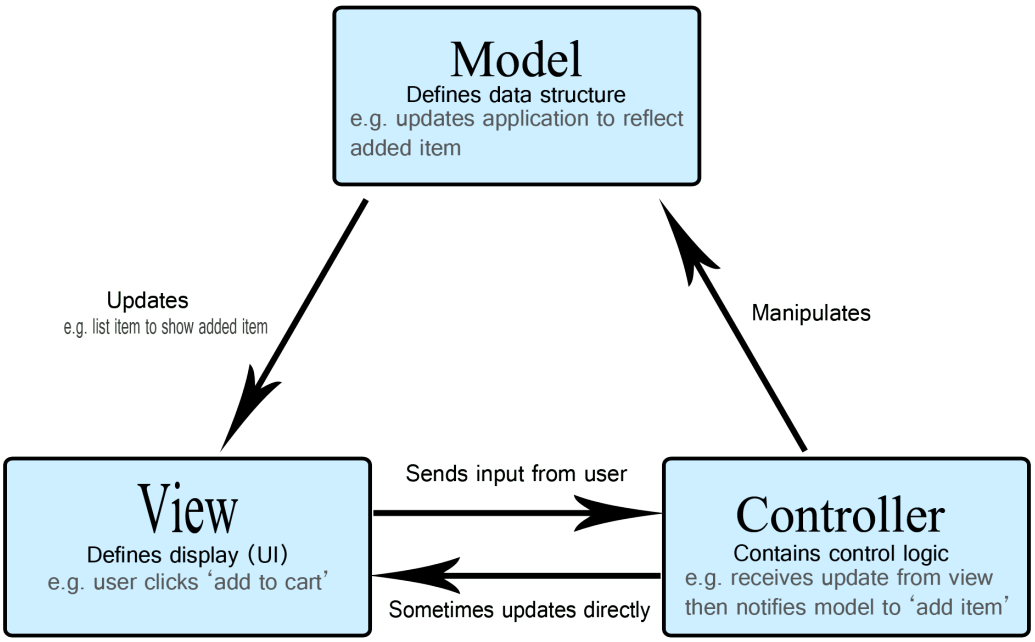

Figura 3.5. Schema grafico del pattern MVC

Django, come mostrato in Figura 3.6., raggruppa il codice che si occupa di ognuno di queste funzionalità in file diversi.

- **URLs:** qui vengono scritte diverse funzioni per gestire ogni URL e quindi ogni risorsa. Un URL mapper viene usato per reindirizzare le richieste HTTP alla vista di competenza basandosi sull'URL della richiesta. Il mapper è in grado anche di abbinare stringhe o numeri presenti nella richiesta e passarli alle funzioni delle views come parametri in input.

- **View:** si tratta di una funzione di gestione della richiesta che riceve la richiesta HTTP e produce in output una risposta HTTP. Le funzioni "views" hanno accesso ai dati per soddisfare la richiesta tramite i modelli e infine delegano la corretta visualizzazione della risposta ai templates.

- **Models:** sono degli oggetti Python che definiscono la struttura dei dati di un'applicazione e forniscono metodi per gestirli nel database.

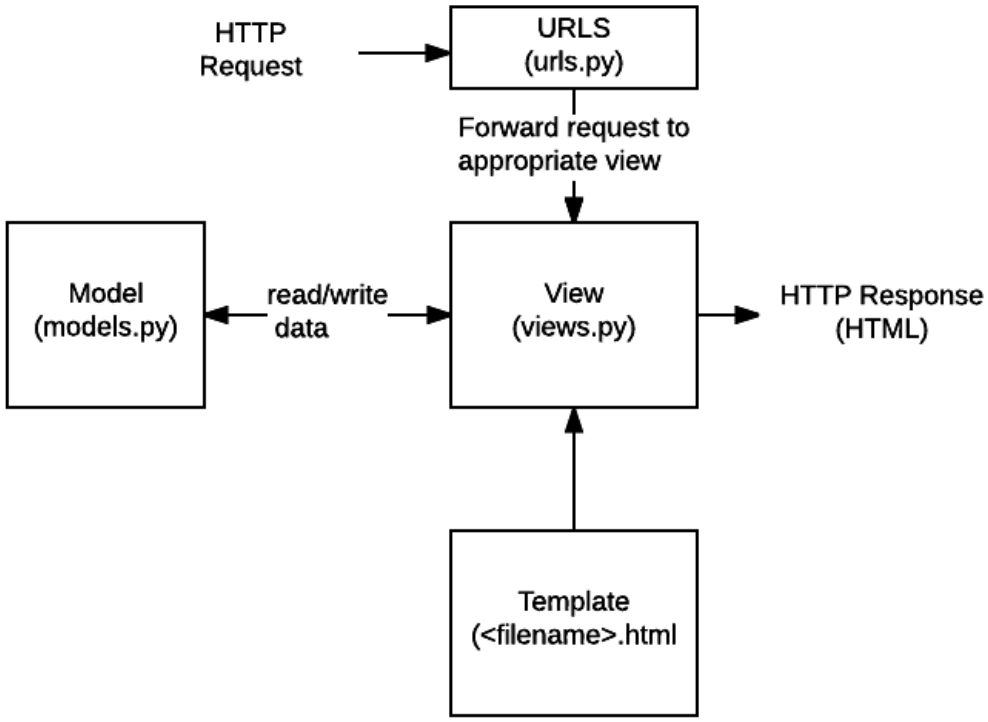

Figura 3.6. Struttura del codice in Django

- **Template:** è un file di test che delinea e stabilisce la struttura o il layout di un file. Una pagina HTML è in grado di essere creata dinamicamente dalla View servendosi di un template HTML e popolandolo con i dati recuperati dal model. In generale un template non deve necessariamente essere un file HTML, può essere usato per descrivere la struttura di qualsiasi file.

3.2.2 Database

Il **database**[1] è un insieme di informazioni o dati strutturati tipicamente archiviati in un computer. Un database è solitamente controllato dal **DBMS**: database management system. Insieme, i dati e il DBMS, insieme all'applicazione alla quale sono legati sono definiti database.

[1] definizioni e informazioni sono fornite dalla guida ufficiale "https://www.oracle.com/database/what-is-database/"

I dati, nella maggiorparte delle operazioni eseguite sui database, sono strutturati in righe e colonne in una serie di tabelle per aumentare l'efficienza di interrogazione. Essi possono essere facilmente modificati, gestiti, aggiornati e cancellati. La maggiorparte dei database usa **SQL** come linguaggio di **interrogazione**.
Un'interrogazione o **query** è una domanda o richiesta espressa con un linguaggio formale [13]. Le interrogazioni di solito si usano per trovare un dato specifico filtrando per qualche criterio. In un database relazionale, l'istruzione SELECT permette all'utente di ricavare un dato dal database verso l'applicazione. Tale risultato viene strutturato come una tabella che viene chiamato **result-set**. L'istruzione SELECT può essere sia arricchita da altre istruzioni come per esempio FROM, WHERE e ORDER BY sia usata per aggregare i dati da analizzare.
Di seguito vengono elencati i principali tipi di database precisando che non c'è il migliore o peggiore in assoluto ma dipende da come vogliono essere usati e organizzati i dati:

- **Relazionali:** gli elementi sono organizzati in tabelle con righe e colonne. Metodo migliore dal punto di vista dell'efficenza e flessibilità se si tratta di dati ben strutturati. Ogni riga della tabella possiede un identificativo univoco chiamato **chiave**. Le colonne invece ospitano gli attributi dei dati.

- **Orientati agli oggetti:** i dati e le informazioni sono organizzati in oggetti, come avviene nei linguaggi di programmazione ad oggetti.

- **Distribuiti:** i dati sono archiviati in due o più file situati in siti diversi. Il database potrebbe essere ospitato da più computer nello stesso sito o anch'essi in siti diversi.

- **Data Warehouse:** si tratta di un database specifico per ottenere interrogazioni e analisi veloci. I dati sono tutti situati in un grande "magazzino" da qui il nome warehouse.

- **NoSQL:** detti anche non relazionali ospitano dei dati non strutturati o semi strutturati. Il loro avvento coincide con l'aumento dello sviluppo web.

- **Cloud:** sono dei database situati in una piattaforma di cloud computing che sia privata, pubblica o ibrida. Archiviano dati sia strutturati e non. Ne esistono di due tipi: tradizionali e DBaaS (database as a service)

dove è un service provider ad eseguire i compiti amministrativi e la manutenzione del database.

MySQL

Il database usato per immagazzinare i dati relativi all'applicazione mobile sviluppata è MySQL. Si tratta di un database management system relazionale sviluppato da Oracle basato sul linguaggio di interrogazione strutturato SQL. I punti di forza di MySQL sono i seguenti:

- **Adatto per sistemi client/server o embedded:** il software di MySQL è composto da un server multi threads che supporta vari backend, librerie e molte API. MySQL viene anche offerto come libreria multithreading per i sistemi embdedded, facile da collegare e usare per migliorare la performance di progetti autonomi.

- **Open source:** chiunque può scaricare, usare e modificare il software di MySQL senza pagare poichè usa la licenza GPL.

- **Affidabile e facile da usare:** le prestazioni di MySQL sono eccellenti quasi in qualsiasi hardware che sia desktop o portatile. Il server di MySQL offre un ricco insieme di funzioni di utilità, inoltre il suo alto livello di sicurezza e connettività lo rende ottimo per accedere ai database sparsi per la rete.

- **Standardizzato:** usa SQL, linguaggio standard per interrogare il database.

- **Schema:** lo schema **rigido** usato da questo sistema relazionale è definito dai vincoli e dalle relazioni tra le tabelle che compongono il sistema. Sono delle regole molto rigide che controllano le operazioni tradizionali dei sistemi RDBMS.

3.2.3 Alternative per il backend

L'obiettivo di questo paragafo è quello di offrire al lettore un'alternativa al backend e database utilizzati nello sviluppo del progetto oggetto di questa tesi.

Alternativa a Django - PHP

PHP è un linguaggio di scripting server-side usato anche per lo sviluppo web. Il nome è un acronimo ricorsivo della definizione inglese **hypertext preprocessor**. La popolarità di questo linguaggio è data dai suoi pregi quali:

- **Facilità d'uso:** la sua sintassi essenziale lo rende un linguaggio facile da imparare sia dai novizi sia da chi già programma in altri linguaggi.
- **Cross-platform:** PHP è indipendente dalla piattaforma sottostante e quindi non è necessario disporre di un sistema operativo specifico.
- **Community:** la community di PHP è attiva ed estesa, per qualsiasi dubbio o quesito è molto probabile che sia stata già pubblicata una risposta.
- **Open source:** il codice sorgente di PHP è disponibile e libero per chiunque voglia sviluppare librerie esterne.

Di seguito invece vengono esposte le principali differenze tra Django e PHP in alcune categorie:

- **Sintassi:** i progetti in Django possono essere esplorati ed eseguiti usando il prompt dei comandi sia di Windows che di linux mentre PHP nonostante abbia una sintassi semplice e intuitiva, il motore di analisi (parsing engine) di PHP deve differenziare il codice normale e quello di PHP.
- **Sicurezza:** PHP dispone di tutte le primitive per rendere sicuro un sito web ma non sono così immediate e facili da usare motivo per cui è necessaria una certa esperienza nel campo. Al contrario Django dispone di un framework con meccanismo automatico per la sicurezza.
- **Caricamento file:** Django si occupa del caricamento file tramite una libreria in python, al contrario in PHP i file sono caricati in via temporanea per poi essere gestiti da uno script che li reindirizza verso la destinazione corretta.

In definitiva la scelta ricade su Django se: il progetto richiede più funzionalità con un tempo di sviluppo ridotto; si vuole sviluppare un'applicazione in grado di essere supportata da molteplici piattaforme o se si vuole usufruire

di tecnologie di intelligenza artificiale.
PHP invece è la scelta da fare se: si vuole un sito web scalabile in grado di integrare un sistema di cloud ma anche un'applicazione web che non necessita dei cambi in tempo reale delle sue caratteristiche principali.

Alternativa a MySQL - database NoSQL

Esistono diverse categorie di database, come descritto nel paragrafo 3.2.2, in grado di gestire diversi tipi di dati. Una di queste categorie nello specifico è quella dei database NoSQL. É un approccio che fornisce uno schema flessibile per l'archiviazione e il recupero dei dati. Un database NoSQL può gestire i seguenti tipi di dati non strutturati:

1. **Chiave-valore:** ogni dato possiede una chiava ed un valore. La chiave è rappresentata tramite stringa, il valore invece è una sequenza di byte non trasparenti al database. Non c'è un linguaggio specifico per interrogare questa tipologia di dati.

2. **Documenti:** i dati sono strutturati in oggetti molto simili ai JSON. Ogni documento contiene coppie di campi e valori. I database con dati organizzati in documenti scalano orizzontalmente e possono ospitare grosse quantità di dati.

3. **Grafi:** ogni struttura dati contiene vertici e archi dove i vertici contengono informazioni sui dati stessi e gli archi rappresentano le relazioni tra i vertici.

4. **Colonne:** sono dati che permettono un accesso molto veloce grazie ad una chiave grezza, il nome della colonna e il timestamp. Archiviati in una tabella, ogni riga non deve necessariamente avere lo stesso numero di colonne.

In definitiva i database NoSQL sono la scelta giusta da adottare per memorizzare grandi quantità di dati non strutturati la cui struttura cambia, per sviluppi e crescita rapida e quando c'è bisogno di uno sviluppo ibrido.

Capitolo 4

Architettura

L'obiettivo principale di questo capitolo è esplicitare e analizzare l'architettura del progetto sviluppato in tutta la sua interezza. Nella prima parte verrà presentata l'applicazione mobile insieme alle schermate principali che la compongono con relativa spiegazione delle funzionalità. Successivamente verrà descritto il principio che governa la comunicazione tra l'applicazione mobile e il backend realizzato in Django come ampiamente discusso nel capitolo precedente. Nella seconda parte invece verranno discusse le scelte dietro la modellazione dei dati e contestualmente presentato il modello di tali dati.

4.1 Analisi funzionale dell'applicazione

L'analisi e la presentazione dell'applicazione verrà eseguita funzionalmente in base ai componenti (widgets) principali che ne fanno parte.

Login / Registrazione

Il componente che si occupa della visualizzazione della schermata di login o registrazione, la prima che vedrà l'utente dopo aver aperto l'app, è nominato `welcome_screen` ed è composto dai seguenti elementi grafici come esposto in **Figura 4.1.** per la registrazione, in **Figura 4.2.** per il login :

- Il logo con il nome dell'applicazione
- Il form dove inserire i propri dati: username e password per il login; username, email, password e la conferma della password per la registrazione

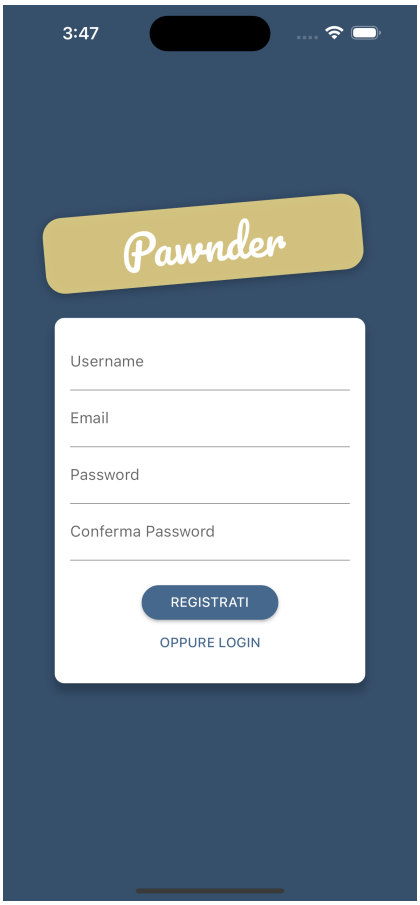

Figura 4.1. Schermata di registrazione

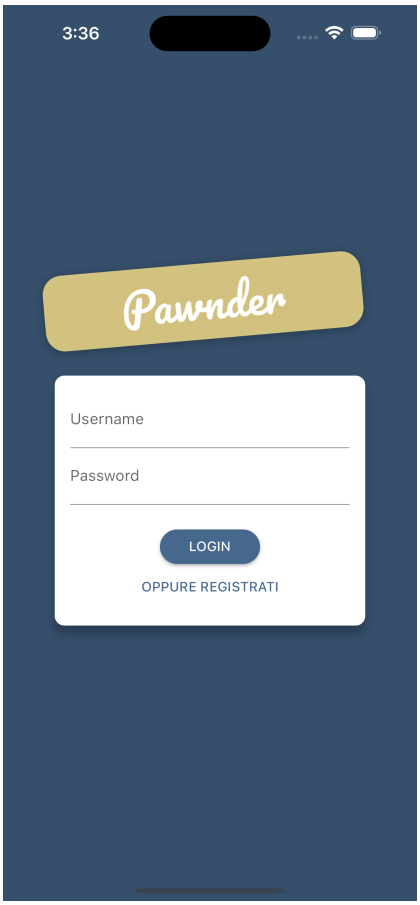

Figura 4.2. Schermata di login

- La CTA "Login" o "Registrati" che porta nella schermata successiva a seconda che ci sia già registrati in app o meno
- La CTA adibita al cambio dinamico del form che permette di visualizzare quello dedicato sia al login sia alla registrazione.

L'utente non registrato, al tap su "Registrati" viene trasferito nella schermata di registrazione gestita dal componente `welcome_screen` composto dai seguenti elementi visibili in **Figura 4.3.** :

- Navbar contenente il titolo della sezione
- Progress bar indicante lo step attuale
- Testo di benvenuto più logo con il nome dell'applicazione

- Testo descrittivo dello scopo dell'applicazione
- CTA "Completa la Registrazione"

Figura 4.3. Schermata di benvenuto

Figura 4.4. Schermata di completamento della registrazione

Al tap sulla CTA "Completa la Registrazione" l'utente vedrà un ulteriore form, gestito dal componente `registration_screen` dove completerà il processo di registrazione inserendo altri dati personali come si vede dalla **Figura 4.4.**, composto dai seguenti elementi:

- Navbar contentente il titolo
- Progress bar indicante lo step attuale
- Testo informativo della schermata

- Form di completamento della registrazione diviso in due parti: prima parte formata dai campi "Nome", "Cognome", "Data di nascita" selezionabile tramite datepicker, "Città", "Via", "N° civico", "Cap"; seconda parte con testo informativo riguardante la tipologia di utente, campi "Regione" e "Nome Associazione".
- CTA "Conferma"

Questionario

Al tap sulla CTA "Conferma" della schermata di registrazione si atterra nella schermata introduttiva gestita dal componente flutter `welcome_survey_screen` e riguarda il questionario che dovrà essere compilato dall'utente in tutta la sua interezza. La Figura 4.5. mostra tale schermata composta come segue:

- Navbar contentente il titolo
- Progress bar indicante lo step attuale
- Testo informativo di introduzione al questionario
- CTA "Vai al questionario"

Dopo aver fatto tap sulla CTA "Vai al questionario" l'utente sarà reindirizzato alla schermata riguardante, pilotata dal componente funzionale `survey_screen`. In tale schermata la domanda sarà visualizzata dinamicamente. Essendo la domanda l'unico elemento mutevole, verrà mostrata solo una schermata esemplificativa che può essere osservata in **Figura 4.6.** contenente:

- Navbar contentente il titolo
- Testo della domanda
- Slider interattivo con valori da 1 a 5 con i quali si risponde alla domanda

Alla fine del questionario verrà presentata una schermata, sempre gestita dal componente `suvey_screen` che indicherà la terminazione dello stesso. Si notano in **Figura 4.7** i seguenti elementi grafici:

- Navbar contentente il titolo
- Testo informativo circa la fine del questionario
- CTA "Vai alla home"

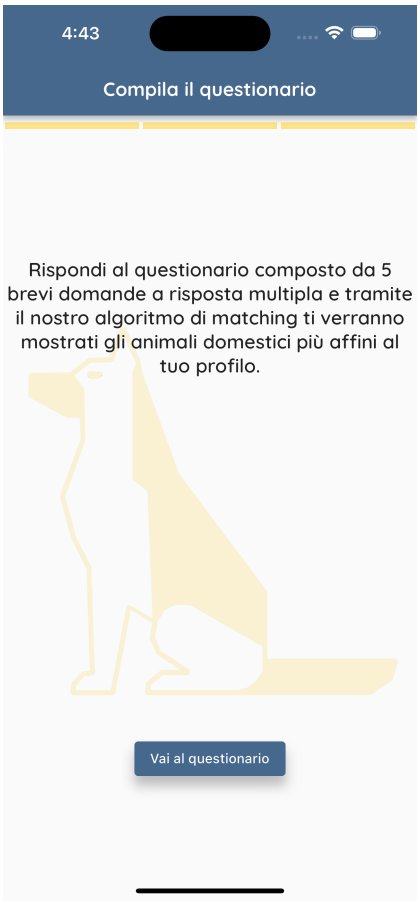

Figura 4.5. Schermata introduttiva al questionario

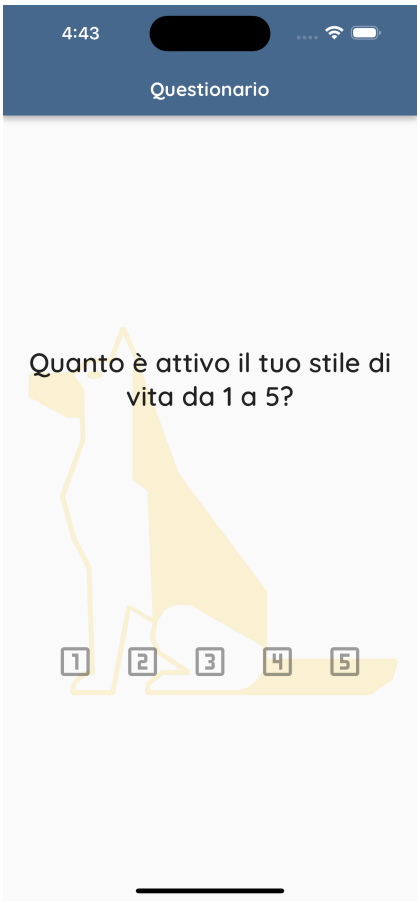

Figura 4.6. Schermata raffigurante una domanda del questionario

Home

Al tap sulla CTA "Vai alla Home" (Figura 4.7) l'utente verrà reindirizzato sulla home page dell'applicazione gestita dal componente `pets_overview_screen`. La schermata può essere raggiunta anche dal flusso di login dopo aver fatto tap sulla CTA "Login" nella schermata mostrata in Figura 4.2. In **Figura 4.8** invece si può notare la schermata home strutturata come segue:

- Navbar contentente il titolo e a destra CTA con icona utente
- Scheda informativa dell'animale domestico che implementa tutte le logiche di rifiuto o accettazione dell'animale domestico tramite swipe a sinistra o destra successivamente nell'estensione. É così composta:
  - Sezione in alto a sinistra con l'anagrafica comprendente nome e età

Figura 4.7. Schermata raffigurante la fine del questionario

Figura 4.8. Schermata della home

  - Informazione sull'affinità[1] tra l'animale domestico e l'utente in alto a destra
  - Sezione in basso a destra con il nome dell'associazione nella quale si trova l'animale domestico

- Estensione della scheda informativa adibita all'espressione di una preferenza con le seguenti CTA:
  - Icona "X" a sinistra per scartare l'animale (stesso effetto di swipe a sinistra)

[1]L'affinità è stata calcolata con la tecnica euristica del coseno di similitudine

  - Icona "i" al centro che reindirizza l'utente alla schermata relativa alle informazioni estese dell'animale domestico
  - Icona cuore a destra per esprimere preferenza di adozione dell'animale domestico.

Dettagli animale domestico

Al tap sulla CTA con l'icona "i" si accede alla sezione dettagli di un animale domestico gestita dal componente flutter `pets_detail_screen`. In **Figura 4.9.** si vedono i dettagli di questa schermata:

- Navbar contentente lo chevron "<" per tornare alla schermata precedente e il nome dell'animale domestico
- Header contenente il titolo "Informazioni generali"
- Generalità dell'animale domestico che includono:
  - Nome
  - Razza
  - Data di nascita nel formato GG-MM-AAAA
  - Associazione
  - Nome
- Header contenente il titolo "Caratteristiche"
- Lista delle caratteristiche dell'animale. Ogni elemento deve includere:
  - Nome della caratteristica
  - Punteggio da 0 a 5 rappresentato da stelle

Profilo utente

La sezione personale dell'utente pilotata dal componente `profile_categories_screen`, raggiunta dal tap sull'icona utente dalla schermata home, presenta delle differenze in quanto gli elementi visualizzati differiscono da utenti dipendenti di rifugi e/o associazioni e non. La **Figura 4.10** e **Figura 4.11** mostrano i seguenti elementi e le differenze:

Figura 4.9. Schermata informazioni estese di un animale domestico

- Navbar contentente lo chevron "<" per tornare alla schermata precedente e lo username dell'utente autenticato
- Griglia con gli elementi che rappresentano le varie categorie delle funzionalità. Ogni elemento contiene l'icona e sotto il testo raffigurante la funzionalità tra cui:
  - "Ripeti il test" che riporta alla schermata del sondaggio in Figura 4.5.
  - "I miei preferiti": sezione con la lista degli animali domestici "piaciuti" dall'utente
  - "I meno votati": schermata contenente la lista degli animali domestici meno votati dagli utenti del sistema

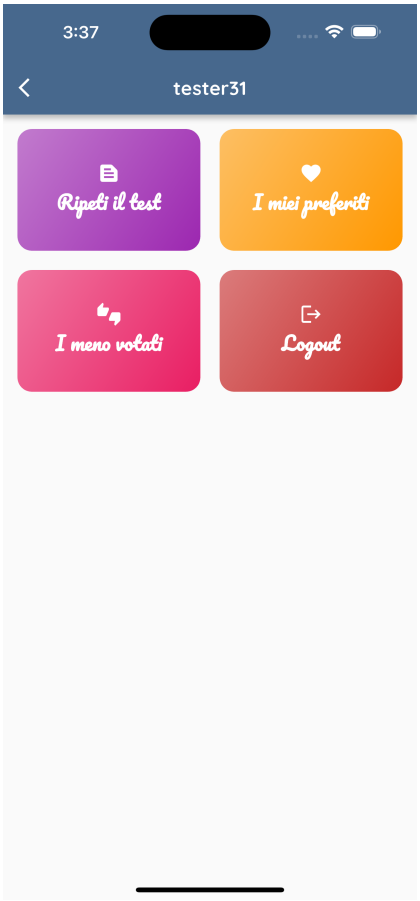

Figura 4.10. Schermata profilo per utenti

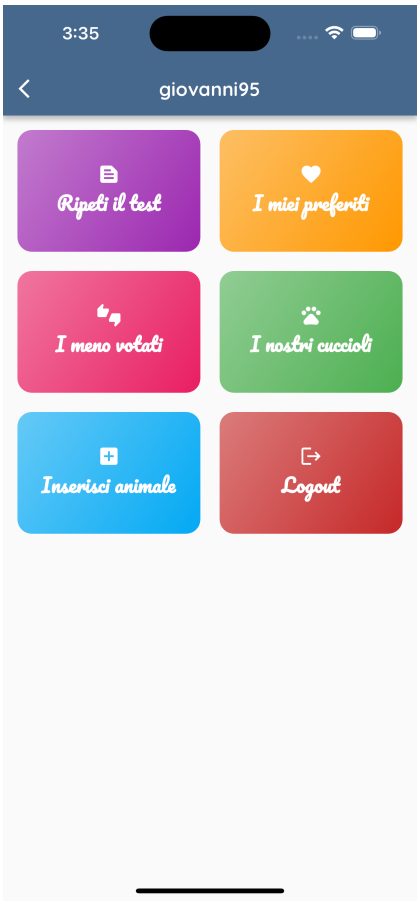

Figura 4.11. Schermata profilo per dipendenti

– "I nostri cuccioli": sezione disponibile solo per i dipendenti, si possono vedere tutti gli animali che alloggiano nell'associazione di cui il dipendente fa parte
– "Inserisci animale": sezione disponibile solo per i dipendenti, funzionalità per aggiungere un nuovo animale domestico nel sistema.
– "Logout": effettua il logout dall'applicazione con reindirizzamento alla pagina di login (Figura 4.2.)

Le sezioni "I miei preferiti", "I meno votati" e "I nostri cuccioli" sono state realizzate tutte tramite lo stesso componente `pet_list_screen`. Ogni schermata, mostrata rispettivamente in Figura 4.12, Figura 4.13 e Figura 4.14, mostra tale componente al variare della funzionalità richiesta. Gli elementi grafici osservabili sono i seguenti:

- Navbar contentente lo chevron "<" per tornare alla schermata precedente e lo username dell'utente autenticato
- Lista degli animali domestici. A seconda della sezione ogni elemento della lista presenta le seguenti caratteristiche:

  1. **I miei preferiti:** icona circolare raffigurante la foto dell'animale, Nome e sotto razza dell'animale, CTA icona "i" che porta alla sezione informazioni (Figura 4.9)

  2. **I meno votati:** icona circolare raffigurante la foto dell'animale, Nome e sotto razza dell'animale, CTA icona "i" che porta alla sezione informazioni (Figura 4.9), CTA icona cuore per esprimere una preferenza positiva, badge con icona e contatore indicante il numero di preferenze ricevute

  3. **I nostri cuccioli:** icona circolare raffigurante la foto dell'animale, Nome e sotto razza dell'animale, CTA icona matita per modificarne le informazioni (Figura 4.1x)

Inserimento / modifica animale domestico

Quando il dipendente fa tap sulla CTA "Inserisci un animale" oppure fa tap sull'icona della matita nella riga di un animale domestico dalla sezione "I nostri cuccioli" viene reindirizzato alla pagina relativamente di inserimento o modifica pilotata dal componente `edit_pet_screen` strutturato come in **Figura 4.15.** :

- Navbar contentente lo chevron "<" per tornare alla schermata precedente e il testo "Info generali"
- Form di inserimento dati composto da i seguenti campi:
  - Nome
  - Razza
  - Note
  - Data di nascita nel formato GG/MM/AAAA da inserire tramite date picker
  - Switch con testo "Nascita esatta?" e relativo toggle

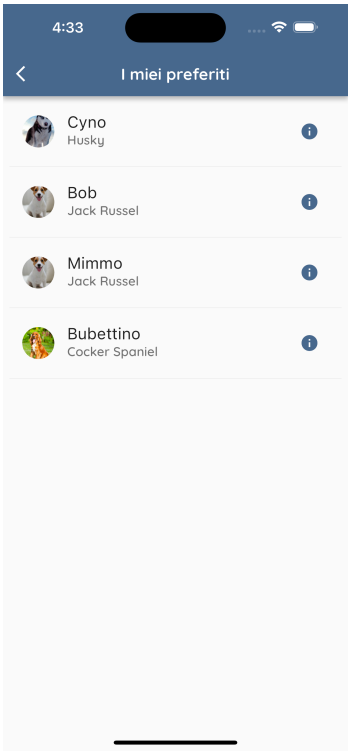

Figura 4.12. Sezione "I miei preferiti"

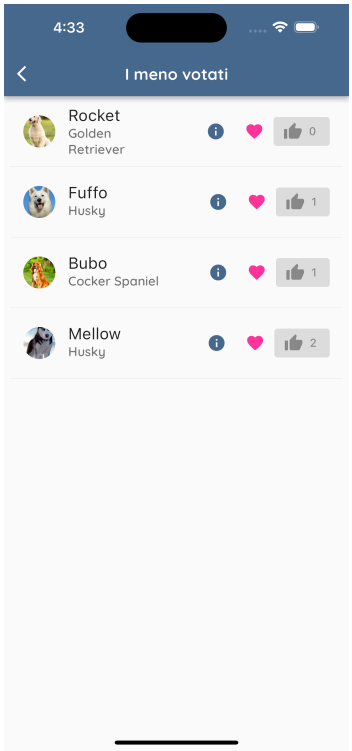

Figura 4.13. Sezione "I meno votati"

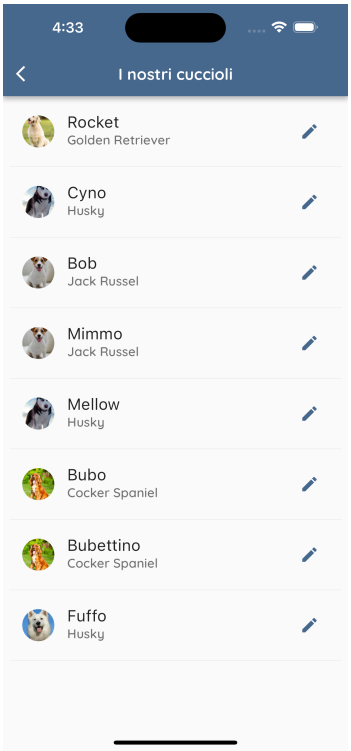

Figura 4.14. Sezione "I nostri cuccioli"

  - Data di adozione
  - Box che raffigura l'anteprima dell'immagine
  - CTA con icona "Scatta una foto" e CTA con icona "Scegli una foto"
- CTA "Avanti"

La seconda parte del processo di inserimento / modifica consiste nell'inserimento o update delle caratteristiche di un animale. Il componente è, in entrambi i casi, `edit_pet_traits_screen` ed è visibile in **Figura 4.16.** con le seguenti caratteristiche:

- Navbar contentente lo chevron "<" per tornare alla schermata precedente e il testo "Caratteristiche"
- Testo "Scorri o tocca sulle stelle per modificare"
- Lista delle caratteristiche presenti nel database. Ogni elemento è strutturato così:

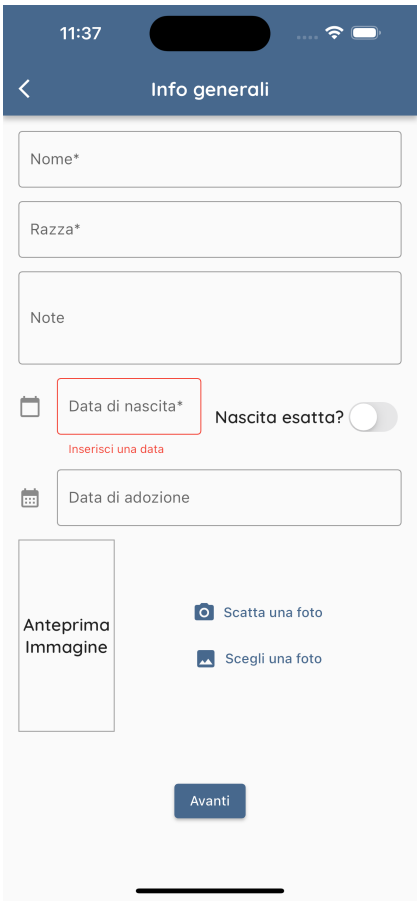

Figura 4.15. Schermata inserimento o modifica animale

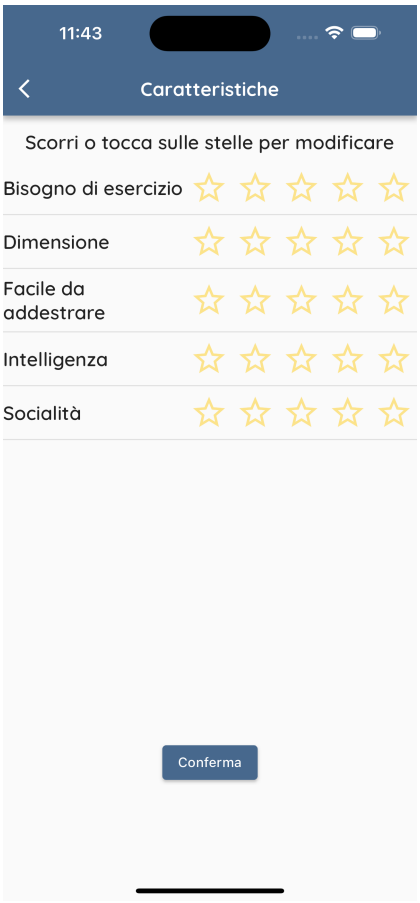

Figura 4.16. Schermata inserimento o modifica caratteristiche animale

  - Nome della caratteristica
  - Slider interattivo con 5 stelle con le quali si valorizza la caratteristica. Sono ammessi anche valori intermedi.

- CTA "Conferma" che termina il flusso reindirizzando l'utente nuovamente nella home.

4.2 Modello degli oggetti - Lista dei widget principali

Nella **tabella 4.1.** sono elencati i widget principali dell'applicazione mobile e la loro funzionalità

Widget	Classe	Funzionalità
AuthCard	auth_screen.dart	Componente che si occupa di permettere all'utente di loggare o registrarsi nell'applicazione tramite un form.
RegistrationScreen	registration_screen.dart	Componente volto a costruire il form di registrazione di un utente.
PetCard	pet_card.dart	Componente che si occupa di visualizzare sotto forma di scheda le informazioni principali di un animale
PetTrait	pet_trait.dart	Componente adibito alla visualizzazione e alla modifica delle caratteristiche di un animale domestico.
CategoryItem	category_item.dart	Componente che genera dinamicamente una voce della sezione profilo e ne gestisce la logica
SurveyItem	survey_item.dart	Componente contenitore, formato da altri componenti minori che insieme realizzano la domanda con le risposte.

Tabella 4.1. Widget principali

4.3 Interazione tra client mobile e backend

Le applicazioni mobile spesso hanno bisogno di interagire con altre applicazioni, o come in questo caso backend, situate in un altro dominio o altre porte. Per stabilire questa interazione è necessario adottare il **CORS** (cross-origin resource sharing) nel server [14]. Per implementare il CORS è stato usato il modulo omonimo di Django.

4.3.1 CORS

Si tratta di un meccanismo che permette di richiedere o più in generale gestire risorse che risiedono in domini diversi da quello in cui servono. Nel caso pratico dell'applicazione web oggetto di questa tesi: il client mobile gira in locale mentre il server Django si trova all'indirizzo `127.0.0.1:8000` per quanto riguarda i device simulati iOS e all'indirizzo `10.0.2.2:8000`. Quando il client fa una richiesta al server Django esso valuta la richiesta basandosi sulla sorgente e sugli headers. Dal momento che nel server sono state esplicitamente permesse le richieste dal client, il server stesso darà la risposta corretta e coerente alla richiesta effettuata, in caso contrario il server darebbe un messaggio di errore. Le informazioni appena citate vengono scambiate tramite le intestazioni HTTP. In **Figura 4.17**, è sintetizzato e schematizzato il meccanismo CORS [15].

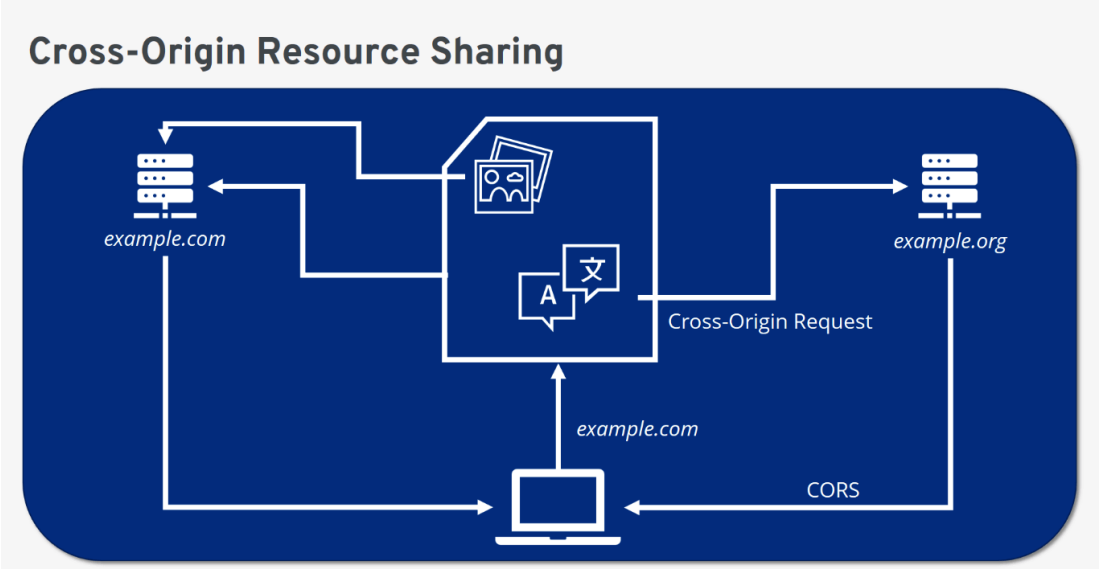

Figura 4.17. Architettura CORS

4.3.2 Abilitare CORS su Django

Di seguito vengono illustrati gli step da seguire per abilitare il modulo CORS su Django:

- **Installare il modulo:** lanciare il comando `python -m pip install django-cors-headers` e inserire in `INSTALLED_APPS` in settings.py il nome del modulo

- **Aggiungere il middleware:** si aggancia alla richiesta / risposta del server, si tratta di un plugin in grado di modificare l'input o l'output di Django. Va inserito in `MIDDLEWARE` sempre in settings.py

- **Definire le impostazioni:** in base alle impostazioni specificate, il modulo decide se la fonte dalla quale viene la richiesta è valida per continuare a processarla al fine di produrre una risposta. Il requisito fondamentale è abilitare il dominio del proprio client nella lista `CORS_ALLOWED_ORIGINS` in settings.py. In opposizione invece, il parametro booleano `CORS_ALLOW_ALL_ORIGINS`, se valorizzato a true permetterà richieste da qualsiasi orgine: scelta sconsigliata in quanto per motivi di sicurezza è buona pratica avere controllo sulle richieste da domini specifici.

4.4 Modello del Database

La modellazione dei dati è stato un processo volto a illustrare i tipi dei dati immagazzinati nel sistema, le relazioni tra essi e le varie combinazioni con le quali tali dati possono essere aggregati, i loro valori e formati [16]. I modelli dei dati possono essere divisi in tre categorie differenti: concettuale, logica, fisica. Di seguito verrà analizzata la prima categoria e successivamente presentata graficamente.
Il modello **concettuale** dei dati è ottima per avere una visione d'insieme del sistema, di cosa conterrà e di come sarà organizzato. Di solito questo modello include le classi delle entità, le loro caratteristiche e vincoli e relazioni.

4.4.1 Diagramma ER

Il diagramma ER (entità-relazione) illustra le relazioni tra entità all'interno di un sistema. Questa tipologia di diagramma è spesso utilizzata per progettare database relazionali grazie ad un insieme di simboli che riesce a catturare graficamente l'interconnessione tra le entità, le relazioni e gli attributi.
Un'**entità** è una "cosa" definibile come una persona, oggetto o evento. La loro forma, nel diagramma ER è il rettangolo.

Una **relazione** rappresenta come le entità si comportano quando interagiscono l'una con l'altra. Di solito la loro forma è il rombo.
La **cardinalità** definisce i valori numerici della relazione tra due entità e possono essere *uno ad uno*, *uno a molti* e *molti a molti*. In **Figura 4.18** è mostrato il diagramma ER dei dati coinvolti nello sviluppo della piattaforma mobile.

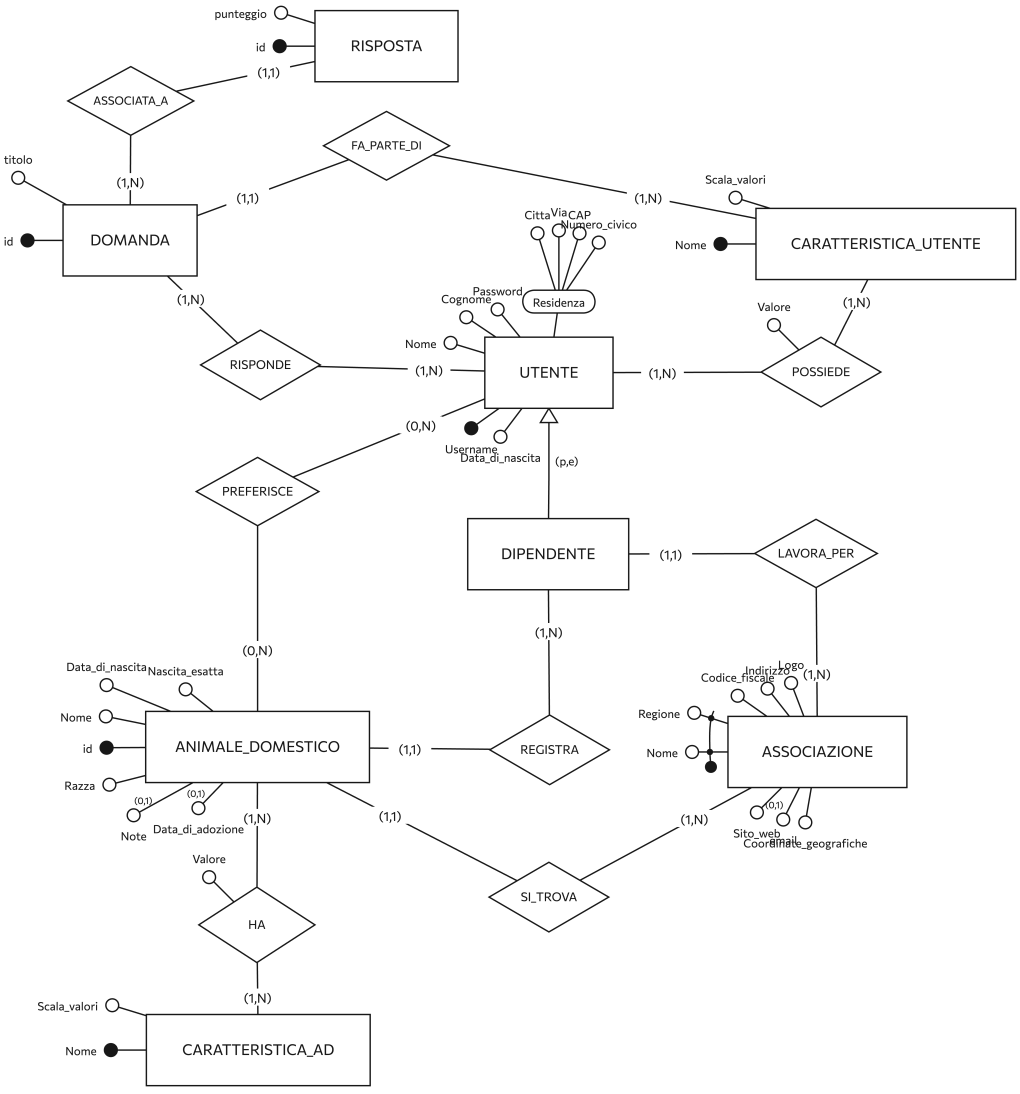

Figura 4.18. Diagramma ER della piattaforma mobile sviluppata

La piattaforma permette di registrare Utenti. Ciascun **Utente** è identificato da un codice univoco, un nome, un cognome, una password, uno username, una data di nascita e da una residenza composta da città, via, CAP e numero civico. Ogni utente *Possiede* una o più **caratteristiche utente** ognuna delle quali è composta da nome e dalla scala di valori. Un Utente può specializzarsi in un **Dipendente** che *lavora per* una o più Associazioni. Un'**Associazione** è composta da un nome e una regione univoci, un codice fiscale, un indirizzo, un logo, un sito web e delle coordinate geografiche. Inoltre un Dipendente può registrare uno o più animali domestici. Un **animale domestico** è identificato da un codice univoco, un nome, una data di nascita, un flag che permette di identificarlo come nato in un giorno specifico o meno, una razza, delle note e un'eventuale data di adozione. Un animale domestico *Si Trova* solo in un'Associazione alla volta. Ciascun animale domestico *Ha* una o più **caratteristiche** identificate dal nome univoco e dalla scala di valori. Un Utente o un Dipdendente può esprimere una *preferenza* per zero o più animali domestici. La piattaforma è anche in grado di registrare un questionario formato da **domande** e **risposte**. Un utente *Risponde* ad una o più domande identificate da un codice univoco e un titolo. Ogni domanda *fa parte di* una o più caratteristica utente ma è anche *associata a* una risposta composta da un identificativo univoco e un punteggio.

Vincoli di tupla

I **vincoli di tupla** o integrità sono condizioni applicate ad ogni tupla della tabella di riferimento del database relazionale. Il vincolo è espresso da un attributo booleano che assume il valore vero se l'istanza rispetta il vincolo, falso in caso contrario. Di seguito sono elencati i vincoli di tupla specifici al diagramma ER mostrato in Figura 4.18. :

- Ogni attributo delle varie classi sarà vincolato in dimensione e tipo.
- Un animale domestico può essere registrato solo da un dipendente
- Un animale domestico non può trovarsi in più associazioni diverse
- Un animale domestico non può essere registrato da più utenti diversi

Capitolo 5

Casi d'uso

Nel capitolo precedente è stata illustrata l'architettura dell'intera piattaforma, sia client mobile che backend, ma soprattutto l'interfaccia utente. In questo capitolo invece, il focus si rivolge sui flussi che possono essere attraversati quando si naviga in app, più nello specifico i **casi d'uso**. Si tratta di descrizioni testuali di come l'utente eseguirà e userà le principali funzionalità dell'applicazione. Verrà descritto, dal punto di vista dell'utente, il comportamento del sistema e come risponderà alle richieste effettuate. Ogni caso d'uso è rappresentato da una sequenza di passi che iniziano con l'obiettivo dell'utente e finiscono col raggiungimento di tale obiettivo. In breve sono delle descrizioni del sistema che fungono da manuale di istruzioni.

5.1 UML & Diagrammi di casi d'uso

UML o Unified Modeling Language è un linguaggio di modellazione standardizzato. Comprende una serie di diagrammi volti ad specificare, documentare e visualizzare i modelli di un sistema software includendo la sua struttura e progettazione. I tipi di diagrammi, come mostrato in **Figura 5.1.** sono principalmente raggruppabili in due categorie: strutturali e comportamentali.

I diagrammi **strutturali** evidenziano la struttura statica del sistema e i suoi componenti su vari livelli di astrazioni e come interagiscono tra loro. I diagrammi **comportamentali** invece mostrano il comportamento dinamico degli oggetti del sistema che possono essere descritti come una serie di cambiamenti nel tempo. In questo capitolo ci si focalizzerà sui diagrammi di casi d'uso.

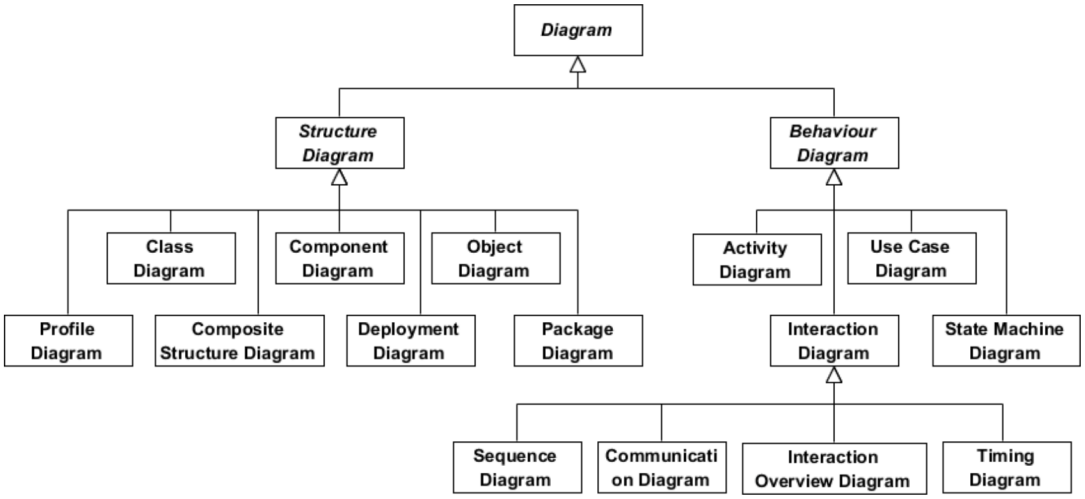

Figura 5.1. Casi d'uso principali

Un diagramma di casi d'uso o **use case diagram** evidenzia i requisiti funzionali del sistema tramite i casi d'uso. Nella notazione grafica gli *ovali* rappresentano i casi d'uso, gli attori invece sono caratterizzati da omini stilizzati e rappresentano gli utenti del sistema con un ruolo specifico ognuno. Le relazioni, rappresentate graficamente da linee rette, mostrano l'intervento di un attore in un caso d'uso.

5.2 Casi d'uso principali

In **Figura 5.2** sono visibili i casi d'uso principali del sistema. Alcuni di essi sono di alto livello e verranno approfonditi ulteriormente successivamente.

- **EffettuaAutenticazione:** caso d'uso che permette la corretta autenticazione degli utenti che intendono utilizzare il software proposto.
- **VisualizzaAnimale:** caso d'uso che permette la lettura delle informazioni generali e delle caratteristiche dell'animale.
- **EffettuaQuestionario:** caso d'uso che permette di effettuare, per la prima volta o volte successive, il questionario proposto dal sistema.
- **ValutaAnimale:** caso d'uso che permette di valutare positivamente o negativamente un animale.

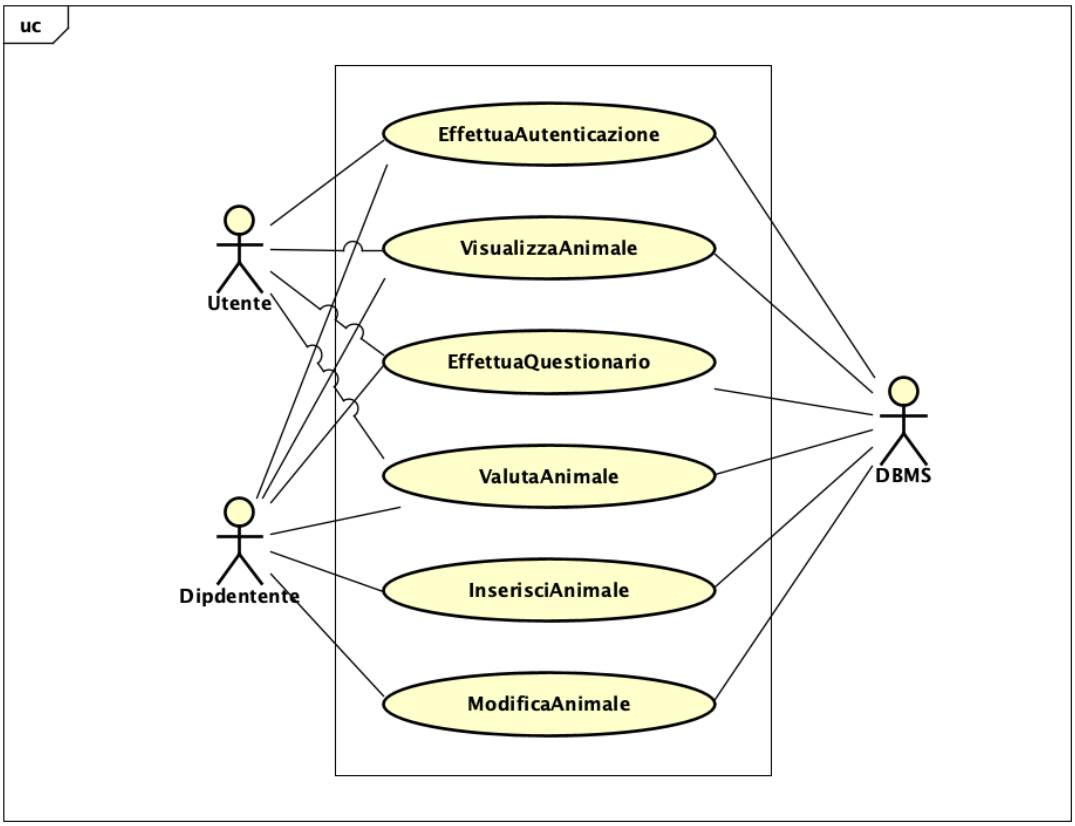

Figura 5.2. Casi d'uso principali

- **InserisciAnimale:** caso d'uso che permette l'inserimento di un animale e di relativi dettagli riguardanti le sue informazioni generali e caratteristiche.

- **ModificaAnimale:** caso d'uso che permette di modificare le informazioni e caratteristiche di un animale

5.2.1 EffettuaAutenticazione

Il caso d'uso EffettuaAutenticazione è un caso composto che può essere diviso in sotto casi d'uso assestanti come mostrato in **Figura 5.3.**

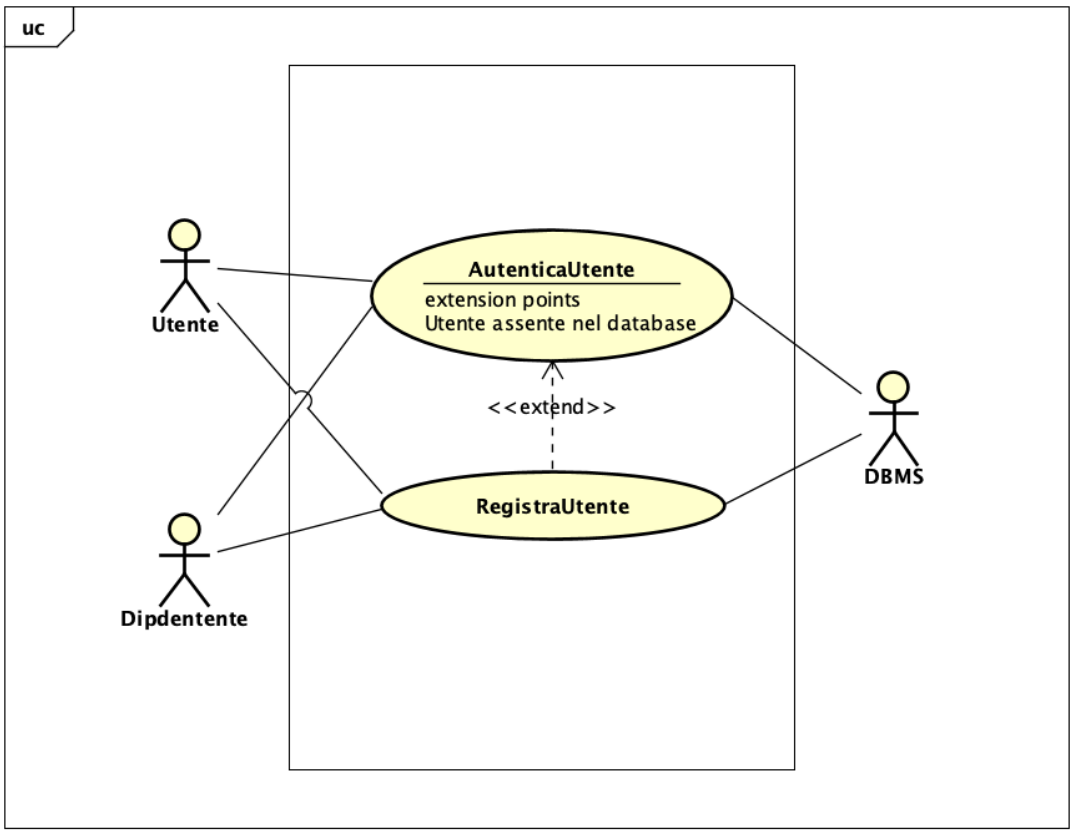

Figura 5.3. Caso d'uso EffettuaAutenticazione

AutenticaUtente

Attori: Utente, Dipendente, DBMS
Precondizioni: L'Utente o Dipendente non si è ancora autenticato in questa sessione
Flusso degli eventi:

1. Il caso d'uso inizia quando un utente[1] avvia il software.

2. Il sistema chiede all'utente di inserire l'username e password.

3. L'utente inserisce il proprio username, la sua password e conferma.

[1]Da qui in poi, utente con l'iniziale minuscola si riferisce ad un generico utente e non all'attore UML Utente

4. Il sistema chiede al DBMS informazioni sull'utente individuato univocamente attraverso l'username che abbia password corrispondente a quella inserita.

5. Il DBMS comunica al sistema se l'utente è presente nel database ed eventuali informazioni connesse.

6. Se l'utente non è presente nel database il sistema informa che l'username o la password sono errati e chiede di reinserirli.

Postcondizioni: Il sistema è nella schermata principale "Home". (Figura 4.8)

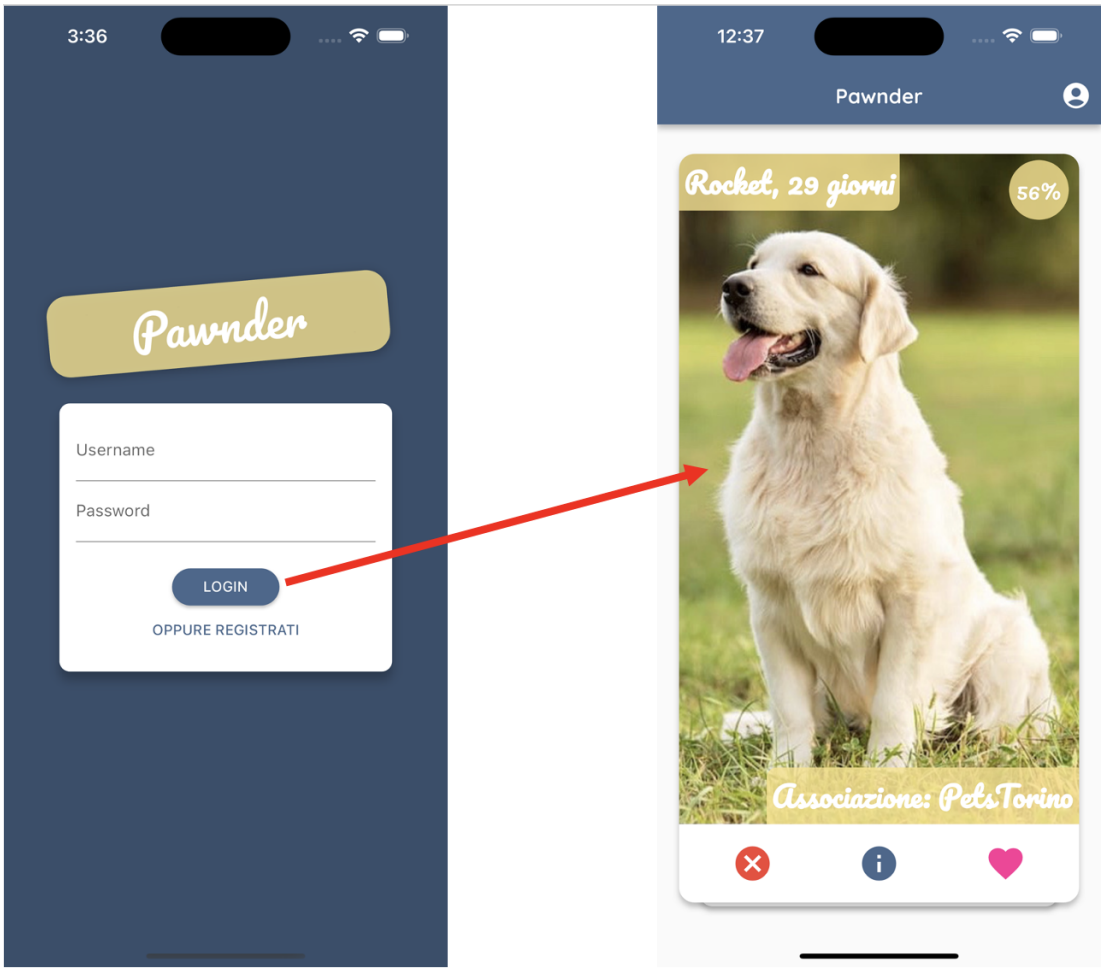

Figura 5.4. Schema logico del caso d'uso AutenticaUtente

RegistraUtente

Attori: Utente, Dipendente, DBMS
Precondizioni: Il sistema è nella schermata di login per Utente o Dipendente (Figura 4.1.)
Flusso degli eventi:

1. Il caso d'uso inizia quando Utente o Dipendente selezionano l'opzione "Registrati" dopo aver inserito username, email e password.

2. Il sistema mostra un ulteriore form per inserire i dati necessari all'inserimento nel sistema

3. L'utente compila il modulo, eventualmente non riempiendo i campi facoltativi e invia i dati.

4. Il DBMS restituisce i dati richiesti.

5. Se l'Utente o Dipdente è già presente nel database viene mostrato un messaggio di errore e viene chiesto di inserire dei dati corretti
   ALTRIMENTI
   Viene comunicato al DBMS di aggiungere il nuovo Utente o Dipendente.

Postcondizioni: Il sistema è nella schermata introduttiva al questionario (Figura 4.5.)

5.2.2 VisualizzaAnimale

Attori: Utente, Dipendente, DBMS
Precondizioni: L'Utente o Dipendente si trova nella schermata principale (Figura 4.8.) o nelle schermate "I miei preferiti" (Figura 4.12) / "I meno votati" (Figura 4.13)
Flusso degli eventi:

1. Il caso d'uso inizia quando l'utente seleziona l'icona "i" nella scheda dell'animale domestico.

2. Il sistema chiede al DBMS di recuperare i dati relativi alle informazioni generali e alle caratteristiche dell'animale domestico per il quale si stanno chiedendo tali informazioni.

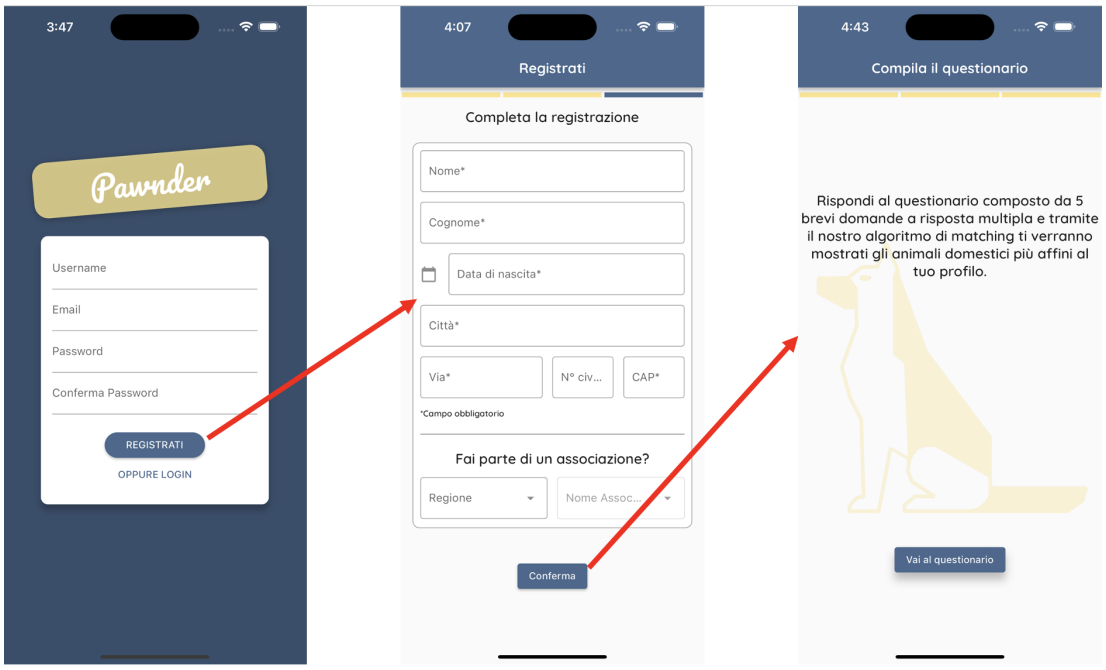

Figura 5.5. Schema logico del caso d'uso RegistraUtente

3. Il DBMS restituisce i dati richiesti.

4. Il sistema mostra in una schermata le informazioni generali e le caratteristiche dell'animale domestico.

Flusso Alternativo:

1. Il caso d'uso inizia quando il Dipendente seleziona l'icona "matita" in corrispondenza di un elemento della lista di animali mostrata dalla schermata.

2. Il caso d'uso continua dal punto 2 come nel flusso principale

Postcondizioni: L'Utente o Dipendente si trova nella schermata di informazioni generali e caratteristiche dell'animale domestico (Figura 4.8)

5.2.3 EffettuaQuestionario

Attori: Utente, Dipendente, DBMS
Precondizioni: L'Utente o Dipendente si trova nella schermata introduttiva

al questionario (Figura 4.5.)
Flusso degli eventi:

1. Il caso d'uso inizia quando l'utente seleziona "Vai al questionario" oppure quando seleziona "Ripeti il questionario" dalla schermata del profilo

2. Il sistema chiede al DBMS di recuperare i dati relativi ad ogni domanda con le relative risposte.

3. Il DBMS restituisce i dati richiesti.

4. Il sistema mostra in una schermata, per ogni domanda, il testo della domanda e le risposte.

5. L'utente risponde alle domande proposte dal sistema e infine seleziona "Vai alla home"

Postcondizioni: L'Utente o Dipendente si trova nella schermata principale "Home" (Figura 4.8.)

5.2.4 ValutaAnimale

Attori: Utente, Dipendente, DBMS
Precondizioni: L'Utente o Dipendente si trova nella schermata principale "Home" (Figura 4.8.)
Flusso degli eventi:

1. Il caso d'uso inizia quando l'utente seleziona l'icona "cuore" nella scheda dell'animale domestico

2. Il sistema comunica al DBMS che l'utente ha valutato positivamente l'animale domestico.

3. Il DBMS aggiorna i dati.

4. Il sistema mostra l'animale domestico successivo.

Flusso Alternativo:

1. Il caso d'uso inizia quando l'utente seleziona l'icona "X" nella scheda dell'animale domestico

2. Il sistema mostra l'animale domestico successivo a seguito della valutazione negativa da parte dell'utente.

Postcondizioni: L'Utente o Dipendente rimane nella schermata principale "Home (Figura 4.8.)

5.2.5 InserisciAnimale

Attori: Dipendente, DBMS
Precondizioni: Il Dipendente si trova nella schermata del profilo (Figura 4.11)
Flusso degli eventi:

1. Il caso d'uso inizia quando il Dipendente seleziona "Inserisci animale" nella schermata.

2. Il sistema mostra un form per inserire i dati necessari all'inserimento di un animale nel sistema.

3. Il Dipendente compila il modulo, eventualmente non riempendo i campi facoltativi e invia i dati.

4. Il sistema chiede al DBMS di aggiungere il nuovo animale domestico.

5. Il DBMS restituisce i dati richiesti.

6. Il sistema mostra una schermata dove inserire le caratteristiche dell'animale.

7. Il Dipendente inserisce un punteggio per ogni caratteristica e invia i dati selezionando "Conferma"

8. Viene comunicato al DBMS di aggiungere le caratteristiche all'animale domestico appena inserito.

Postcondizioni: Il dipendente si trova nella schermata "Home" (Figura 4.8.)

5.2.6 ModificaAnimale

Attori: Dipendente, DBMS
Precondizioni: Il Dipendente si trova nella schermata del profilo "I nostri cuccioli" (Figura 4.14)
Flusso degli eventi:

1. Il caso d'uso inizia quando il Dipendente seleziona l'icona "matita" in corrispondenza di un elemento della lista di animali mostrata dalla schermata.

2. Il sistema chiede al DBMS di recuperare i dati relativi all'animale domesitco selezionato.

3. Il DBMS restituisce i dati richiesti.

4. Il sistema da inizio al caso d'uso InserisciAnimale

Postcondizioni: Il dipendente si trova nella schermata "Home" (Figura 4.8.)

5.3 REST API server

In questo paragrafo verranno elencate tutte le API effettuate dal sistema verso il DBMS e raggruppate in categorie apposite in base alla loro funzionalità.

5.3.1 Gestione Utente

In questa categoria rientrano le API che si occupano della corretta registrazione dell'utente così come del login e del logout.

- POST `api/login/`
  - Contenuto di body nella richiesta: username e password
  - Contenuto di body nella risposta: oggetto utente valorizzato
- POST `api/logout/`
  - Contenuto di body nella richiesta: oggetto utente valorizzato
  - Contenuto di body nella risposta: vuoto
- POST `api/register/`

  - Contenuto di body nella richiesta: username, password e email
  - Contenuto di body nella risposta: oggetto utente valorizzato

5.3.2 Risorse Utente

In questa sezione sono state raggruppate tutte le API sotto il path `users`, responsabili dell'interazione tra Utente e DBMS

- POST `api/users/<str:username>/edit/`
  - Parametri nella richiesta: username dell'utente
  - Contenuto di body nella richiesta: oggetto utente valorizzato
  - Contenuto di body nella risposta: oggetto utente valorizzato
- POST `api/users/<str:username>/like/<int:pk>/`
  - Parametri nella richiesta: username dell'utente, id dell'animale domestico
  - Contenuto di body nella richiesta: vuoto
  - Contenuto di body nella risposta: vuoto
- GET `api/users/<str:username>/liked/`
  - Parametri nella richiesta: username dell'utente
  - Contenuto di body nella risposta: lista di animali domestici valutati positivamente dall'utente
- POST `api/users/<str:username>/answers/`
  - Parametri nella richiesta: username dell'utente
  - Contenuto di body nella richiesta: lista di risposte
  - Contenuto di body nella risposta: vuoto
- POST `api/users/<str:username>/traits/`
  - Parametri nella richiesta: username dell'utente
  - Contenuto di body nella richiesta: lista di risposte
  - Contenuto di body nella risposta: vuoto

5.3.3 Gestione Animali Domestici

In questa categoria sono racchiuse le API sotto il path `pets`. Si tratta di API volte al recupero e corretta gestione dell'animale domestico.

- GET `api/pets/<str:username>`
  - Parametri nella richiesta: username dell'utente
  - Contenuto di body nella risposta: lista di animali domestici
- POST `api/pets/add/`
  - Contenuto di body nella richiesta: oggetto animale domestico da inserire
  - Contenuto di body nella risposta: oggetto animale domestico inserito
- GET `api/pets/lessliked/`
  - Parametri nella richiesta: nessuno
  - Contenuto di body nella risposta: lista di animali domestici meno votati
- GET `api/pets/<int:pk>/`
  - Parametri nella richiesta: id dell'animale domestico
  - Contenuto di body nella risposta: oggetto animale domestico valorizzato
- POST `api/pets/<int:pk>/update/`
  - Parametri nella richiesta: id dell'animale domestico
  - Contenuto di body nella richiesta: oggetto animale domestico da aggiornare
  - Contenuto di body nella risposta: oggetto animale domestico aggiornato
- GET `api/pets/<int:pk>/traits/`
  - Parametri nella richiesta: id dell'animale domestico
  - Contenuto di body nella richiesta: vuoto

  - Contenuto di body nella risposta: lista di caratteristiche dell'animale domestico

- POST `api/pets/<int:pk>/traits/`
  - Parametri nella richiesta: id dell'animale domestico
  - Contenuto di body nella richiesta: caratteristiche da aggiornare dell'animale domestico
  - Contenuto di body nella risposta: vuoto

- GET `api/pets/<str:regione>/<str:associazione>/`
  - Parametri nella richiesta: regione e nome dell'associazione
  - Contenuto di body nella risposta: lista di animali domestici in affido all'associazione il cui nome è stato passato in input

5.3.4 Extra

Negli Extra rientrano tutte le API volte a fornire al sistema tutte le informazioni aggiuntive richieste per un corretto funzionamento.

- GET `api/regioni/`
  - Parametri nella richiesta: nessuno
  - Contenuto di body nella risposta: tutte le regioni presenti nel database

- GET `api/questions/`
  - Parametri nella richiesta: nessuno
  - Contenuto di body nella risposta: tutte le domande presenti nel database

- GET `api/traits/`
  - Parametri nella richiesta: nessuno
  - Contenuto di body nella risposta: lista di caratteristiche presenti nel database

- GET `api/associations/<str:region>`

- Parametri nella richiesta: regione
- Contenuto di body nella risposta: lista di associazioni presenti nella regione passata in input

Capitolo 6

Conclusione & Sviluppi Futuri

La piattaforma sviluppata è sicuramente un mezzo da tenere in considerazione per abbassare il tasso di abbandono degli animali domestici e promuovere, insieme a tutte le campagne e ai consigli dell'ENPA, l'adozione soprattutto da canili, rifugi e associzioni. L'applicazione, prodotto del lavoro di questa Tesi Magistrale, è stata realizzata interamente da zero, in flutter e dart come linguaggio di programmazione sottostante. Sono state usate alcune librerie esistenti del framework Flutter per realizzare alcuni componenti in app. La funzionalità principale, il calcolo dell'affinità tra utente e animale domestico, è stato eseguito sfruttando il coseno di similitudine implementato tramite la libreria numpy in python. Il backend o server è stato realizzato con Django web framework e python come linguaggio di programmazione. Il database scelto è MySQL, database relazionale. Il client mobile e il backend interagiscono tramite il modulo CORS del framework Django. Trattandosi ancora di un prototipo, è migliorabile e in continuo sviluppo con la possibilità di aggiungere nuove funzionalità tra cui: **potenziare il recommendation system** usato impiegando l'uso di reti neurali o altre tecniche discusse nel Capitolo 2; implementare un sistema di **geolocalizzazione** in grado di fornire un suggerimento migliore e tracciare eventuali spostamenti degli animali tra rifugi; inserire una sezione dedicata volta ad integrare la piattaforma con **corsi di addestrmento** e/o **pet therapy**; possibilità di caricare, contestualmente a foto anche eventuali **video**, aggiungere alla sezione profilo una pagina dedicata a **filtri** per ottimizzare la ricerca; offrire supporto anche per **altre tipologie** di animali domestici come gatti e conigli.

Infine, sotto quest'ottica, in vista di un futuro rilascio negli App Store e Google Play Store, verrebbero chieste convenzioni e collaborazioni con i principali canili e rifugi nel territorio Italiano.

Bibliografia

[1] Redazione (23 Agosto 2022) *Abbandono degli Animali in Estate: un triste fenomeno.* Robinson pet Blog. https://www.robinsonpetshop.it/news/cane/abbandono-animali-cosa-fare-per-soccorrere/

[2] Pham, K. (2022, July 12). *What are Recommendation Systems?*. Medium. https://medium.com/@khang.pham.exxact/what-are-recommendation-systems-6bb5036042db

[3] *Recommendation systems* (n.d.) Engati. https://www.engati.com/glossary/recommendation-systems

[4] Michael D. Ekstrand, John T. Riedl and Joseph A. Konstan (2011), *"Collaborative Filtering Recommender Systems"*, Foundations and Trends® in Human–Computer Interaction: Vol. 4: No. 2, pp 81-173. http://dx.doi.org/10.1561/1100000009

[5] Aggarwal, C.C. (2016). *Content-Based Recommender Systems.* In: Recommender Systems. Springer, Cham. https://doi.org/10.1007/978-3-319-29659-3_4

[6] Iateilang Ryngksai, L. Chameikho, 2014, *Recommender Systems: Types of Filtering Techniques*, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) Volume 03, Issue 11 (November 2014)

[7] Shiu-li Huang, *Designing utility-based recommender systems for e-commerce: Evaluation of preference-elicitation methods*, Electronic Commerce Research and Applications, Volume 10, Issue 4, 2011, Pages 398-407, https://doi.org/10.1016/j.elerap.2010.11.003.

[8] Jansen, S. J. (2011). *The multi-attribute utility method.* The measurement and analysis of housing preference and choice, 101-125.

[9] Safoury, L.A., & Salah, A.I. (2013). *Exploiting User Demographic Attributes for Solving Cold-Start Problem in Recommender System.*

[10] Aggarwal, C.C. (2016). *Knowledge-Based Recommender Systems*. In: Recommender Systems. Springer, Cham. https://doi.org/10.1007/978-3-319-29659-3_5

[11] Burke, R. (2007). *Hybrid Web Recommender Systems.* In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds) The Adaptive Web. Lecture Notes in Computer Science, vol 4321. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-72079-9_12

[12] MDN contributors. (2023) *MVC*, mdn web docs. https://developer.mozilla.org/en-US/docs/Glossary/MVC

[13] Zola, A. & Hughes A. (2021) *query*. TechTarget. https://www.techtarget.com/searchdatamanagement/definition/query

[14] StackHawk (2021, April 30). *Django CORS Guide: What It Is and How to Enable It*. StackHawk. https://www.stackhawk.com/blog/django-cors-guide/

[15] Digital Guide IONOS (2019 December 12). *CORS: illustrazione del Cross-Origin Resource Sharing*. IONOS. https://www.ionos.it/digitalguide/siti-web/programmazione-del-sito-web/cross-origin-resource-sharing/

[16] IBM *What is data modeling?*. IBM. https://www.ibm.com/topics/data-modeling