POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Gestionale

Tesi di Laurea Magistrale

Quando il codice incontra il guinzaglio

Piattaforma web per connettere persone e animali compatibili

Relatore: Prof. Alessandro Fiori

Candidato: Elisabetta Girolami

Anno accademico 2025–2026

Ai miei nonni,

per avermi cresciuto,
sostenuto,
ed insegnato ad amare.

Indice

1 Introduzione 1

1.1 Panorama attuale 1

1.2 Struttura della Tesi 3

2 Recommendation Systems 4

2.1 Tipi di Recommendation System 4

2.1.1 Modelli Collaborative Filtering 6

2.1.2 Modelli Content-Based 7

2.1.3 Modelli Knowledge-Based 8

2.1.4 Altri tipi di sistemi 11

3 Strumenti per lo sviluppo web 14

3.1 Django 14

3.1.1 Architettura MTV 15

3.1.2 ORM 18

3.1.3 Alternative a Django 20

3.2 Docker 24

3.2.1 Componenti principali 24

3.2.2 Docker vs Macchine Virtuali 26
3.2.3 Vantaggi di Docker 28
3.3 Database Relazionali 28
3.3.1 La teoria di Codd 29
3.3.2 Relazionale vs NoSQL 31
3.4 Bootstrap 34
3.4.1 Installazione 34
3.4.2 Componenti 35

4 Tecnologie per implementare il Recommendation System 37
4.1 Architettura 37
4.2 Metriche 39
4.2.1 Cosine Similarity 39
4.2.2 Distanza Euclidea 41
4.2.3 Pearson Correlation Coefficient 42
4.2.4 Jaccard Similarity 44
4.2.5 Mean Squared Difference 45

5 La piattaforma a livello funzionale 47
5.1 Struttura dei dati 47
5.2 Registrazione 50
5.2.1 Registrazione Utente semplice 50
5.2.2 Registrazione Associazione e Proprietario 52
5.2.3 Registrazione Animale 54

5.3 I template del sito 54
5.3.1 Pagine Personali 55
5.3.2 Animali e Associazioni 57
5.3.3 Quiz 59
5.4 Caratteristiche 61
5.4.1 Caratteristiche Animali 62
5.4.2 Caratteristiche Utenti 64

6 Analisi dei risultati 67
6.1 Dati in input 67
6.1.1 Utenti 67
6.1.2 Animali 69
6.1.3 Situazione complessiva in input 70
6.2 Output 71
6.2.1 Utente Uno 71
6.2.2 Utente Due 72
6.2.3 Utente Tre 73
6.3 Analisi delle metriche 74
6.3.1 Distanza Euclidea 74
6.3.2 Cosine Similarity 75
6.3.3 Pearson Correlation Coefficient 76
6.3.4 Jaccard Similarity 77
6.3.5 Mean Squared Difference 78
6.3.6 Confronto delle metriche 80

7 Conclusioni 82

7.1 Valore aggiunto 83

7.2 Sviluppi futuri 83

Bibliografia 84

Capitolo 1

Introduzione

Se è vero che il cane è il miglior amico dell'uomo, non è altrettanto vero il viceversa. Ogni anno, ed in particolar modo ogni estate, troppi di questi "migliori amici a 4 zampe" vengono abbandonati, con una stima, come riporta [1] di 130.000 animali domestici abbandonati ogni anno, solo in Italia.

Questo è il punto di partenza di questa tesi, che ha come sogno, più che come obiettivo, la fine di questo comportamento criminale: l'obiettivo, più realistico, di questo progetto, è quello di una piattaforma che, basandosi su un algoritmo di raccomandazione, cerchi il match perfetto tra un utente ed il suo futuro animale domestico, basandosi su un quiz che vada ad indagare tutti quegli aspetti che, in una relazione tra uomo e fedele amico a 4 zampe, potrebbero costituire motivi di fastidio o problemi e che vada a collegare gli animali alle persone che realmente potrebbero prendersi cura delle loro esigenze. Ovviamente, per realizzare il sogno che sta alla base di tutto ciò, la parte principale spetta poi agli esseri umani: il compito di ritrovare la propria coscienza e non abbandonare un cane, un gatto o una qualsiasi forma di vita per qualunque motivo, poiché non esiste un motivo che giustifichi o legittimi questo comportamento.

1.1 Panorama attuale

Navigando sul web si possono trovare numerosi siti, e anche delle applicazioni, che offrono un supporto alla ricerca dell'animale domestico. Esse fungono principalmente da punto d'incontro tra gli utenti e gli animali a loro vicini, in alcuni casi possono chiedere delle informazioni aggiuntive per filtrare o ricercare figure

adatte, ma puntando tendenzialmente a porre al centro dell'attenzione le esigenze dell'utente, i suoi desideri e le sue comodità.

Un esempio di ciò è costituito dalla piattaforma "Animal Friends - Adopt a dog" [2], che mostra annunci di cani adottabili e ne permette l'adozione tramite la compilazione di un modulo in cui vengono chieste informazioni base dell'utente come indirizzo, città e telefono. Potrebbe esservi un'analisi delle richieste da parte di volontari delle associazioni, questo non è visibile dal sito, ma resterebbe uno studio del caso in questione lasciato in carico al personale, mentre nel sito non sembrano esservi accenni a strumenti che calcolino un'eventuale affinità nè quiz da sottoporre.

Vi è una piattaforma, "PetRescue - Pet Match Quiz" [3], che propone un mini quiz di quattro domande per consigliare l'animale adatto all'utente. Queste domande offrono ciascuna tre opzioni di risposta ed alla fine consigliano la razza e il tipo di animale che ritengono più affine; in particolare chiedono: il motivo dell'adozione (le cui risposte spaziano dal salvare una vita all'avere un compagno da portare ovunque), il passatempo preferito dell'utente (come incontrare gli amici o stare a casa e guardare la TV), che carattere dovrebbe avere l'animale domestico (ad esempio "curioso e giocherellone") ed infine come dovrebbe essere l'animale ideale, offrendo come alternative "Si adagia alle mie ginocchia per le coccole", "È molto amorevole" e "Riempie il mio cuore fino a farlo traboccare", quindi restando sul tema di un animale molto dolce nei confronti del proprio padrone. Da questo emerge in maniera immediata quanto il focus sia posto sull'utente mentre gli animali vengono categorizzati sulla base della propria razza e tipo, offrendo quindi un consiglio finale che indichi quale "tipologia" sarebbe più indicata per le caratteristiche desiderate. Questa piattaforma, se da un lato implementa un breve quiz e può vagamente essere paragonata al progetto di questa tesi, in realtà risulta allontanarsi fortemente dalle idee e dagli obiettivi che ha questo progetto, non analizzando match per match sulla base delle caratteristiche specifiche dell'utente e dell'animale e non ponendo il focus, come dovrebbe essere, sull'animale e sulle sue esigenze. Una volta consigliata la tipologia di animale, "PetRescue" offre la possibilità di visualizzare gli animali registrati su di essa che appartengono a quella categoria e di filtrare i risultati tramite alcune caratteristiche, che mettono sempre in luce i bisogni dell'utente.

Similmente a questa piattaforma vi sono altri siti che offrono una sorta di match tra l'utente che compila il quiz e le varie razze, come "Purina - MatchMe / Breed Selector" [4]. Questo sito offre una gamma di domande già maggiore del precedente e apparentemente focalizzate sia sull'utente che sull'animale: per carti aspetti come il rumore o la richiesta della taglia dell'animale riprende aspetti considerati anche nel progetto di questa tesi. È un ottimo strumento educativo, come mostrato anche dai vari riferimenti ad articoli per la cura ed il benessere dell'animale e offre attraverso il quiz un iniziale approccio dell'utente per comprendere magari

alcuni aspetti rilevanti da considerare quando si desidera adottare un cucciolo. Pur trattandosi di un valido strumento non è completamente paragonabile a questo progetto: infatti non si propone di creare match singoli e di fornire un ponte diretto tra le associazioni e gli utenti, e la generalizzazione per "razze" non permette dei veri e propri match.

Complessivamente quindi, nel panorama attuale non vi sono dei veri e propri competitor di questo progetto, che invece si pone l'obiettivo di essere sia un ponte diretto tra associazioni e utenti, sia di calcolare dei match affidabili sulla base di caratteristiche personali dell'utente e dell'animale in questione, senza generalizzazioni per razza o tipologia. Vi sono però valide piattaforme che condividono lo scopo di questo progetto di educare le persone al rispetto e alla cura degli animali, e da cui anzi prendere spunto per poter implementare magari nuove funzionalità volte a questo.

1.2 Struttura della Tesi

Nei prossimi capitoli viene analizzata la piattaforma che è stata creata, a partire dalle architetture utilizzate fino ad un'analisi dei risultati ottenuti tramite gli algoritmi di raccomandazione.

In particolare nel Capitolo 2 viene svolta una piccola introduzione ai Recommendation Systems e ai differenti modelli in cui si distinguono. Nel Capitolo 3 sono presentati gli strumenti per lo sviluppo web che sono stati impiegati per realizzare questa piattaforma. Nel Capitolo 4 vengono analizzate le tecnologie sfruttate per implementare i sistemi di raccomandazione, con una descrizione dell'implementazione tramite il linguaggio python di ogni metrica all'interno di questo progetto. Nel Capitolo 5 viene fornita una panoramica della piattaforma e del suo funzionamento. Nel Capitolo 6 vengono analizzati i risultati ottenuti dall'inserimento a sistema di tre utenti e cinque animali e viene fornita una panoramica delle cinque metriche di affinità implementate per evidenziare quale di queste si presti meglio al caso in questione. Infine nel Capitolo 7, relativo alle conclusioni, viene ricordato l'obiettivo di questo progetto, spiegato il valore aggiunto che può apportare nel panorama attuale ed ipotizzati gli sviluppi futuri che questa piattaforma potrebbe avere.

Capitolo 2

Recommendation Systems

Un recommendation system è un sistema di raccomandazione, basato sull'analisi di big data ed algoritmi di machine learning, che cerca di trovare degli schemi (ad esempio nel comportamento degli utenti) per suggerire all'utilizzatore i prodotti che potrebbero maggiormente interessargli [5]. L'idea che sta alla base dei sistemi di raccomandazione infatti, è quella di unire ed usare tutti i dati che riescono a reperire, per intercettare gli interessi degli utenti [6]. Questo può avvenire usando diversi tipi di sistemi, da quelli più collaborativi a quelli basati sui contenuti, con tante diverse sfaccettature intermedie.

2.1 Tipi di Recommendation System

Prima di analizzare i vari tipi di sistemi, la prima differenza riguarda la formulazione del problema, che può avvenire tramite due modelli principali[6]:

- Versione predittiva: si ha una sorta di matrice, data da m utenti per n elementi, e si vuole predire il valore che ogni elemento avrebbe per ogni utente, andando a colmare le caselle vuote della matrice.

- Versione di classificazione (ranking): in cui non si ritiene necessario calcolare ogni combinazione utente-elemento, ma si desidera solo ottenere una sorta di ordinamento per poter consigliare all'utente in questione i primi k elementi che per lui potrebbero essere più rilevanti.

Il ranking però potrebbe essere vista, e talvolta è così che viene usata, come un passaggio successivo alla prima formulazione: prima si ottengono i valori assoluti di ogni coppia utente-item, e successivamente vengono ordinati gli items per prenderne i primi k più rilevanti in base all'ordinamento utilizzato.

Generalmente i sistemi di raccomandazione sono fortemente impiegati nell'ambito dell'e-commerce, e vengono sfruttati con l'obiettivo di aumentare le vendite ed i ricavi dell'azienda, ed un esempio paradigmatico di ciò è costituito da Amazon con la sua celebre dicitura "I clienti che hanno acquistato questo articolo hanno comprato anche..." o la sezione ispirata alla propria cronologia di navigazione. Dato l'obiettivo commerciale che viene spesso associato ai recommendation system, Charu C. Aggarwal ha stilato una lista delle principali caratteristica che un sistema di raccomandazione dovrebbe avere[6]:

- Rilevanza: devono consigliare prodotti rilevanti per l'utente o catturare il suo interesse.
- Novità: l'utilità delle raccomandazioni risiede in qualcosa che l'utente non si aspetterebbe e che di conseguenza non avrebbe cercato in autonomia.
- Serendipità: questo concetto si lega molto a quello precedente, ma contiene al suo interno una componente di vera e propria sorpresa, in quanto l'utente non scopre solo prodotti nuovi, cioè che prima non conosceva, ma proprio inattesi e inaspettati, cosa che potrebbe fornirgli un certo entusiasmo nel continuare a visionare gli elementi che gli vengono consigliati, aumentando le probabilità di acquistare un maggior numero di prodotti.
- Diversità delle raccomandazioni: è essenziale per permettere all'utente di comprare più di un prodotto o restare su un sito per più tempo, poiché infatti nessuno comprerebbe lo stesso prodotto più e più volte solo perché gli viene consigliato, o leggerebbe lo stesso articolo per tutto il giorno.

I tipi di dati su cui si basano i sistemi di raccomandazione sono principalmente due: le interazioni tra utente e item, e le informazioni sugli attributi degli utenti e degli elementi. In base a che tipi di dati vengono utilizzati si può già fare una classificazione tra recommendation systems: i modelli collaborative filtering sfruttano le interazioni (come numero di like, numero di click su una pagina, o il fatto di aver effettivamente acquistato un prodotto), mentre i metodi content-based recommender si basano sugli attributi degli elementi che devono raccomandare, cercando di riportare quindi tutto ad un livello più verbale, trasformando il testo in "parole chiave", o il genere e il regista di un film in dei "tag" facili da gestire. Vi sono poi i sistemi di raccomandazione knowledge-based, ovvero basati sulla conoscenza, che concettualmente si avvicinano molto a questi ultimi, ma oltre a differenze che

saranno esaminate più avanti, si basano su requisiti che l'utente è chiamato ad esternare in modo esplicito. Oltre a questi, e molte altre categorie più specifiche che da questi derivano, vi sono poi i sistemi ibridi, che combinano due o più di questi metodi per trovare la combinazione più adatta al caso specifico.

2.1.1 Modelli Collaborative Filtering

I modelli collaborativi, come suggerisce il nome, si basano sulla "collaborazione" tra utenti (o items) e quindi sull'affinità tra due entità diverse ma con gusti simili. Questi modelli sono spesso visti come una generalizzazione dei modelli di regressione, in cui da un numero di variabili definite si deve calcolare la variabile mancante (ovvero la variabile dipendente); questo spiega perché molte volte vengano utilizzati principi della regressione, applicati ai sistemi di raccomandazione. Si può comprendere il funzionamento di questi algoritmi immaginando due utenti, con gusti molto simili in acquisti precedenti: il modello ipotizzerà che anche in futuro possano tendere a comprare elementi simili, e quindi all'esecuzione di un acquisto da parte di uno dei due utenti in questione, potrebbe consigliare lo stesso prodotto anche all'altro. Similmente, come mostrato in Figura 2.1, la "raccomandazione" può avvenire su due articoli da leggere: per cui ritenendo due utenti simili in base ai passati articoli con cui hanno interagito entrambi, la piattaforma potrebbe consigliare articoli letti da uno dei due utenti in questione anche all'altro.

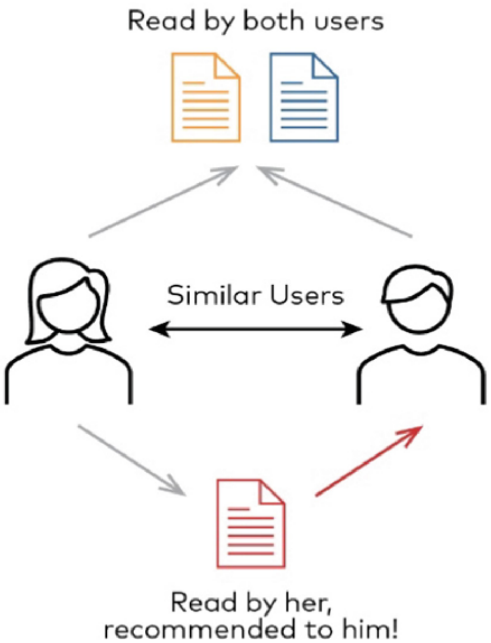


Figura 2.1: Schema del funzionamento del filtraggio collaborativo.

Questo sta alla base della precedentemente citata sezione di Amazon "I clienti che hanno acquistato questo articolo hanno comprato anche..." e può essere visto da due diversi punti di vista, come si vedrà successivamente, cioè associando utenti simili tra loro, oppure prodotti simili tra loro. Questi modelli sono divisi come segue:

- Memory-based methods, di cui fanno parte:
  - User-based collaborative filtering
  - Item-based collaborative filtering
- Model-based methods

Nei metodi memory-based tipicamente l'affinità tra utente e item è calcolata sulla base dei propri "vicini", con una distinzione in base al soggetto su cui viene eseguito questo calcolo. Infatti questa si distingue in filtraggio collaborativo basato sull'utente o sull'elemento. Nel primo caso viene calcolata la somiglianza tra due utenti che accedono alla piattaforma: si considera un utente target di cui si vuole conoscere l'affinità ai prodotti offerti e si cercano tutti gli altri consumatori con gusti simili all'utente target per consigliare gli stessi prodotti che hanno soddisfatto loro. Nel filtraggio item-based, similmente, non entrano in gioco altri utenti ma gli altri items, andando a ricercare tutti i prodotti che sono ritenuti maggiormente simili o affini all'elemento che ha scaturito interesse nel consumatore in questione.

I metodi model-based invece basano tutto su modelli predittivi che sfruttano il machine learning ed il data mining, usando ad esempio alberi di decisione, o modelli rule-based. Sostanzialmente, mentre il filtraggio memory-based studia i dati contenuti a database per effettuare predizioni, i metodi model-based usano quegli stessi dati per creare dei modelli, che poi generino predizioni. Questi hanno una copertura maggiore dei modelli memory-base, ma possono essere più complessi da implementare.

2.1.2 Modelli Content-Based

Nei sistemi di raccomandazione basati sul contenuto vengono usati gli attributi propri degli item per fornire raccomandazioni: infatti il termine "content" prensente nel nome si riferisce alla descrizione propria dell'item in oggetto (il suo contenuto). Tipicamente essi si basano su "parole" (o tag) e modelli di regressione o classificazione. Un esempio paradigmatico di questo tipo di algoritmi è dato dai film, in cui non basta (o non sarebbe un ottimo sistema) vedere quali sono i film più

popolari da consigliare senza tenere in considerazione una serie di variabili, come il genere, il regista, gli attori presenti o le tematiche trattate, che sono tutte difficili, se non impossibili, da esprimere a livello puramente numerico e necessitano invece di "keyword", delle specie di tag che racchiudano al loro interno le "caratteristiche" principali del film (ovvero che esprimano il contenuto del film).

Rispetto ai modelli precedenti quindi, questi algoritmi non ricercano gusti affini tra gli utenti, ma si concentrano su un singolo utente e ricercano similitudini tra gli item da consigliare. Come mostrato in Figura 2.2 infatti, in questo caso il modello parte dagli item che considera graditi dall'utente, come ad esempio un articolo di giornale, e ricerca elementi simili da proporre: ritenendo un articolo simile ad uno già visionato, il modello consiglia all'utente la lettura di questo nuovo elemento.

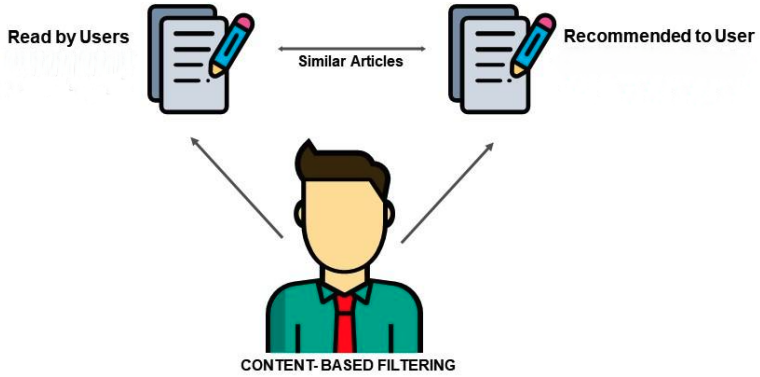


Figura 2.2: Schema del funzionamento dei modelli content-based.

Poiché questi sistemi si basano sul contenuto degli elementi, caratteristica implicita in essi, e non su pareri o voti che debbano arrivare dall'esterno, hanno un grande vantaggio rispetto ai modelli precedenti (collaborative-filtering), ovvero non necessitano di una grande quantità di dati per iniziare ad eseguire delle valutazioni, e riducono di molto il problema del cold start, anche se un minimo input iniziale è necessario per capire i gusti dell'utente e trarre da essi delle "parole chiave" a cui associarlo (basti pensare alla richiesta di Netflix di selezionare i film di proprio gusto quando si crea un nuovo account).

2.1.3 Modelli Knowledge-Based

Come suggerisce il nome, questi modelli si basano sulla conoscenza, intesa in modo molto simile ai modelli precedenti, ma determinando poi una grande quantità di differenze per cui è giusto presentarli separatamente, anche se spesso vengono

considerati due volti della stessa medaglia. I sistemi knowledge-based sono fondamentali quando si tratta di "beni" (items) acquistati poco frequentemente come auto, case ecc, o su cui comunque non vi sono abbastanza interazioni da poter definire un particolare algoritmo di raccomandazione. In questo contesto quindi, partendo sempre da attributi propri degli elementi analizzati, si cerca un'affinità tra questi ed i valori esplicitati dall'utente come target di riferimento. Come mostrato in Figura 2.3, l'utente inserisce semplici input, che vengono elaborati dal sistema grazie alla conoscenza di esperti del settore, e portano alla visualizzazione dei risultati più affini alle richieste dell'utente. Da questo punto in poi vi sono una serie di iterazioni che permettono all'utente di perfezionare i parametri della sua ricerca, visualizzando di volta in volta risultati sempre più attinenti.

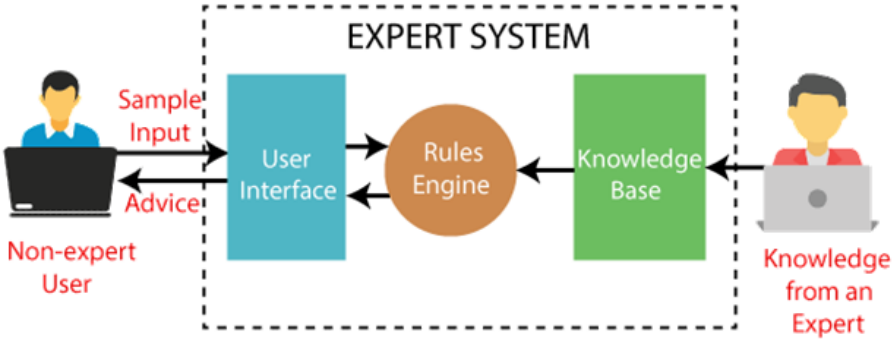


Figura 2.3: Schema del funzionamento dei modelli knowledge-based.

Un esempio paradigmatico di questi modelli è dato da un utente che necessita di comprare una casa: non si tratta di un bene che acquista frequentemente, quindi non ci si può basare su interazioni passate, e per quanto possa avere gusti simili a determinati altri utenti, potrebbe anche avere (quasi sicuramente avrà) necessità differenti e più specifiche relativamente ad una casa in cui andare ad abitare, o potrebbe avere in mente un particolare tipo di alloggio che sogna da tempo. Questo esempio serve a capire i punti caratteristici del modello in questione, infatti per questo caso (come tutti quelli coperti dal medesimo sistema) difficilmente si hanno dati su cui creare dei modelli; al contrario è necessario il più delle volte un input esplicito da parte dell'utente, che si troverà ad esempio ad interagire con la piattaforma per definire un range per ogni caratteristica rilevante oppure inserire i valori della casa dei suoi sogni per trovare quella disponibile più vicina alle sue esigenze. Quest'ultima distinzione è riportata dalle due sottocategorie in cui questi sistemi si dividono:

- Constraint-based recommender systems
- Case-based recommender systems

La prima differenza che emerge tra questi, anche perché si tratta di una differenza principalmente visiva, è il layout dell'interfaccia: i constraint-based, come si può vedere in Figura 2.4, avranno bisogno di un'interfaccia in cui poter specificare un range, fisso e chiaro, per ogni caratteristica dell'item in questione, entro cui si vogliono valutare le opzioni (e quindi "scartare" ogni item che esce da quell'intervallo).

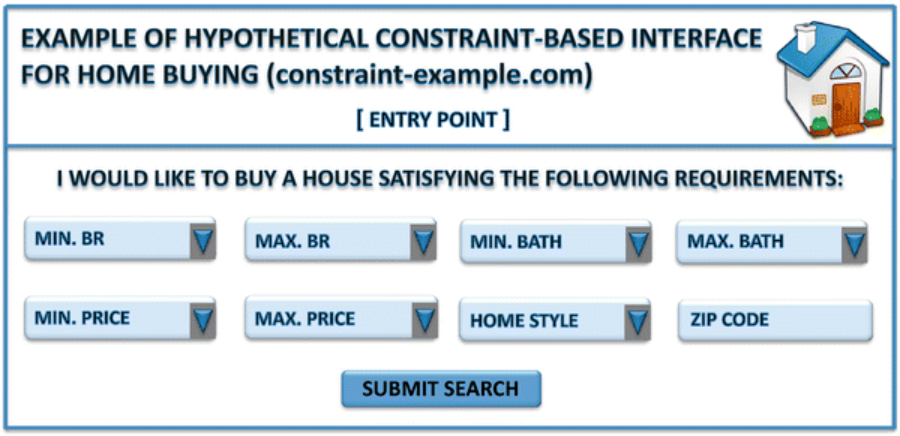


Figura 2.4: Interfaccia di un sito basato sull'approccio constraint-based.

Al contrario nel case-based system, come mostrato in Figura 2.5, gli utenti saranno chiamati a descrivere, fornendo più caratteristiche possibili, l'item (ad esempio la casa) che hanno in mente ed il sistema risponderà fornendogli in output tutti gli elementi più simili trovati.

Un punto in comune ad entrambi è l'iterazione che tipicamente si ha in questi sistemi, in cui l'interfaccia permette di modificare, o in alcuni casi consiglia proprio di farlo tramite suggerimenti, gli input inseriti dall'utente per restringere il campo analizzato fino a trovare l'item desiderato o più adatto alla situazione. Anche in questo caso però vi è una differenza tra le due sotto-categorie, determinata dal modo in cui vengono effettuate queste modifiche: nel caso dei sistemi constrint-based tipicamente si implementa un sistema predisposto a fornire già suggerimenti relativi agli input da modificare, soprattutto per i casi in cui la ricerca è talmente rigida da non portare a nessun risultato, e quindi la piattaforma consiglia quali criteri cambiare o come farlo, oppure il caso contrario in cui si rimane troppo sul vago e vengono restituiti talmente tanti risultati per cui sarebbe impossibile analizzarli tutti. Quindi questi sistemi consigliano modifiche da effettuare nelle iterazioni successive alla prima, e così facendo allargano o restringono il campo o addirittura prendono in considerazione item differenti a seconda del numero di modifiche effettuate. Diversamente, nel caso dei sistemi case-based, una volta forniti i dati che la piattaforma necessita in input, si può interagire con il sistema per fornirgli in tempo reale feedback utili a comprendere quali item restituiti siano realmente di interesse per l'utente o quali attributi di uno specifico elemento siano

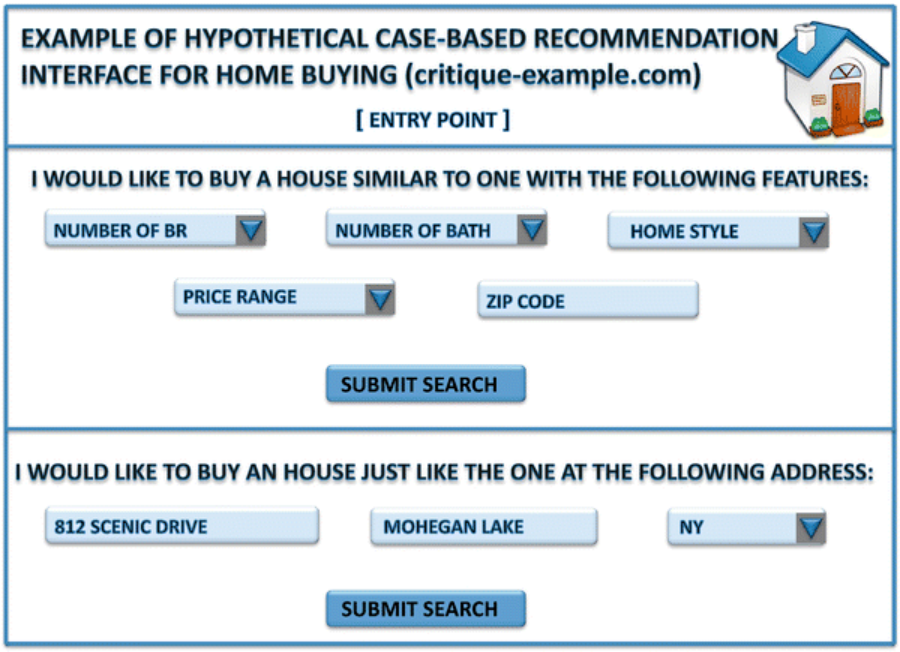


Figura 2.5: Interfaccia di un sito basato sull'approccio case-based.

da modificare, e talvolta anche in che direzione modificare quel valore (se il prezzo debba essere più basso, o le stanze più numerose ecc).

2.1.4 Altri tipi di sistemi

In generale, e soprattutto ad oggi, è riduttivo basarsi solo su un modello unico ed isolato; la pratica attualmente più comune infatti riguarda l'unione di più modelli al fine di ottenere predizioni il più accurate possibili e sfruttando tutti i dati che si possono ricavare, come mostrato in Figura 2.6.

In questo vasto panorama di possibilità sono nati, e si possono continuamente sviluppare, nuovi modelli, che si concentrano in particolare su una caratteristica o su un settore ma che di fatto raramente vengono impiegati come modelli autonomi, più che altro sono implmentati come parte di sistemi ibridi. Qui di seguito viene proposta una breve visione dell'enorme panorama da loro costituito.

Vi sono i modelli Utility-based, in cui si crea una funzione di utilità sulla base delle caratteristiche dell'item, per calcolare la probabilità che ad un utente piaccia un certo elemento. In questi contesti, la sfida riguarda il definire una funzione di utilità appropriata alla situazione, anche perché in questi modelli la funzione di

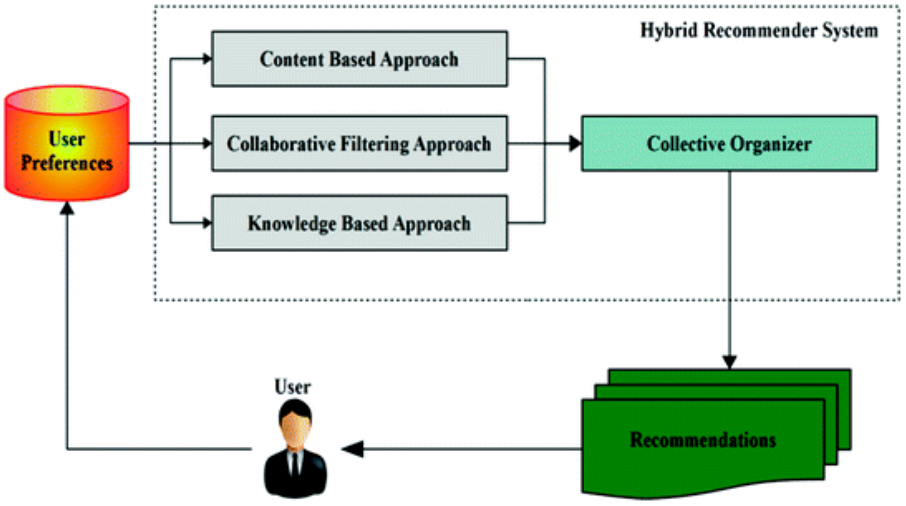


Figura 2.6: Funzionamento di un sistema ibrido.

utilità deve essere conosciuta a priori. Per questa logica, i modelli utility-based spesso sono visti come un caso specifico dei modelli basati sulla conoscenza.

Poi esistono sistemi di raccomandazione demografici, in cui la posizione non viene banalmente usata come filtro, ma assume un ruolo importante per fare assunzioni, e basare su di esse un vero e proprio modello. Nella maggior parte dei casi però, questo tipo di modelli non viene implementato da solo, ma combinato con altre forme per ottenere un sistema il più completo possibile, e la componente demografica, che può assumere un peso rilevante nel calcolo finale, è sfruttata come valore aggiunto.

Relativamente a questo vi sono molti altri sistemi di raccomandazione che, sfruttando particolari tipi di dati, in base agli input disponibili o più significativi per il caso specifico, migliorano notevolmente le raccomandazioni fornite, giocando un ruolo protagonista in sistemi considerati ibridi o ensambled-based (d'insieme). Un esempio di questi casi, oltre al precedentemente visto sistema demografico, è dato dai modelli basati sul contesto, che combinano diverse caratteristiche "contestuali" dell'utente, come la posizione geografica ma non solo: dati sociali, temporali, stagionali ecc. Per comprenderne meglio l'utilità basti pensare ad un sistema di raccomandazioni implementato in un e-commerce di abbigliamento: una componente molto rilevante per le vendite sarebbe sicuremante data dalla temperatura e dalle stagioni del luogo da cui l'utente considerato si collega al sito per effettuare un acquisto, oppure i tipi di abiti che, tramite interazioni sui social o like ad altri contenuti, possono maggiormente rispecchiare i suoi gusti.

Similmente vi sono sistemi di raccomandazione time-sensitive, ovvero sensibili

alla componente temporale, come nei casi in cui un prodotto può acquistare o perdere l'interesse degli utenti al passare del tempo, oppure può attirare maggiormente l'attenzione in determinati periodi, giorni o orari. Un esempio è dato dai film, la cui raccomandazione potrebbe differenziarsi molto a seconda che il film sia appena uscito nelle sale o sia quasi al termine della programmazione. La maggior parte di questi sistemi, ed in particolar modo questo, possono essere ricondotti ai tre principali modelli precedentemente visti, ma qui ognuna delle caratteristiche che fornisce il nome al sistema gioca un ruolo talmente importante da necessitare di essere menzionati per completezza.

Allo stesso modo vanno citati i sistemi location-based, soprattutto in un'epoca in cui le cordinate GPS fornite dagli smartphone giocano un ruolo talmente rilevante da portare una moltitudine di applicazioni a richiederne la condivisione nel momento dell'utilizzo. La geolocalizzazione non riguarda solo la vicinanza a determinati punti di interesse come viene immediato pensare, ma permette di collegarsi in qualche modo ai modelli sopra citati, per comprendere una serie di attributi contestuali dell'utente in questione, caratteristiche che giocano un ruolo fondamentale nel determinarne o predirne le preferenze, e questo è il caso della località specifica dell'utente. Dall'altro lato della medaglia vi è però sicuramente anche la località specifica dell'item (o travel locality), che si riferisce al comune calcolare la distanza tra l'utente e l'elemento considerato per fornire raccomandazioni che siano utili in quanto raggiungibili.

Infine non si possono non menzionare i Social Recommender System, basati su strutture, segnali social e tag. Infatti molti prodotti sono consigliati in base alla loro popolarità social, e questo talvolta diviene anche un problema, conosciuto come "viral marketing" e necessita di un'attenta analisi per verificare l'influenza e gli elementi rilevanti sulla rete, per associare ad esempio a loro un maggior peso ai loro input in quel determinato contesto. Ma questi modelli racchiudono un panorama più vasto di questo, comprendendo ad esempio anche il ranking delle pagine, la classificazione dei link o la classificazione colettiva.

Capitolo 3

Strumenti per lo sviluppo web

Questo capitolo offre una panoramica sugli strumenti per lo sviluppo web utilizzati in questo progetto, con un piccolo confronto con le alternative esistenti. In particolare, come si vedrà, è stato adoperato: il framework Django per il back-end, Docker per simulare l'ambiente di sviluppo e i database relazionali per gestire e organizzare i dati. È stato inoltre utilizzato il framework front-end Bootstrap per rendere il progetto responsive e implmentare lo stile della piattaforma senza perdersi nella scrittura di una vasta gamma di regole CSS.

3.1 Django

Django è un framework web Python di alto livello [7] utilizzato nella realizzazione di siti per velocizzare il processo e rendere l'applicazione più sicura e scalabile. In parole povere si tratta di una sorta di cassetta degli attrezzi, che offre molti strumenti a disposizione dei programmatori, così che possano concentrarsi sulla realizzazione del proprio sito senza riscrivere gli stessi blocchi di codice o dover implementare tutte le funzioni di base[8].

I principali vantaggi di Django sono[9]:

- Completezza: si tratta di un framework "batteries included", ovvero che offre già tutte le componenti di base che possono servire al programmatore, senza che debba replicarle da zero o creare nuove librerie, e questo è strettamente collegato anche con il principio DRY ("don't repeat yourself") incoraggiato da Django, che tenta di eliminare la ridondanza e la duplicazione del codice. Tra

le caratteristiche che lo rendono completo, e non solo, si trova sicuramente il suo ORM (Object Relational Mapping), che permette di trasformare record del database in oggetti (ad esempio istanze Python) senza la necessità di scrivere query SQL[10]. Inoltre trattandosi di un framework open source gode di una grande popolarità e di una comunità di sviluppatori in continua creascita, sempre pronti ad intervenire per migliorarlo e fornire suggerimenti e consigli utili [8].

- Versatilità: questo framework è adatto alla realizzazione di qualsiasi tipologia di sito o social network, e fornisce contenuti in tanti diversi formati, come ad esempio HTML o JSON, ma può essere anche esteso tramite componenti esterne.

- Sicurezza: permette di proteggere il proprio sito dagli attacchi più comuni, come SQL injection e cross-site scripting, attraverso l'uso, da parte dello sviluppatore, di poche e semplici righe di codice.

- Scalabilità: l'architettura di questo framework è di tipo "Shared-nothing" (cioè nulla di condiviso), ovvero un'architettura distribuita, costituita da più nodi indipendenti, ognuno dei quali con una propria memoria, archiviazione ed una potenza di elaborazione [11], ed ognuno che lavora su dati differenti dagli altri. Poiché ogni parte risulta quindi indipendente da tutto il resto, può facilmente essere sostituita o modificata senza problemi o perdite di efficienza ed apre anche le porte alla possibilità di aumentare il traffico qualora fosse necessario. Questo tipo di architettura non solo comporta vantaggi in termini di scalabilità, ma migliora anche le prestazioni evitando colli di bottiglia, una maggiore tolleranza ai guasti ed aumenta l'affidabilità.

- Manutenibilità: basandosi sul principio DRY precedentemente citato, Django incoraggia la creazione di codice manutenibile e riutilizzabile, evitando ripetizioni e riducendo la quantità di righe di codice.

- Portabilità: essendo un framework scritto in Python è quindi funzionante su molte piattaforme e supportato dai più diffusi sistemi operativi.

3.1.1 Architettura MTV

Django utilizza l'architettura MTV (Model-Template-View), che è una particolare versione dell'architettura MVC (Model-View-Controller)[12].

Nell'architettura MVC, la cui struttura è illustrata in Figura 3.1, ogni componente svolge un ruolo specifico: il Model si occupa della struttura dei dati e definisce

ad esempio quali componenti debba contenere una determinata classe; la View definisce come mostrare quei dati e cosa mostrare all'utente; il Controller è colui che prende le decisioni, ovvero ad ogni input dell'utente può aggiornare o modificare model e view.

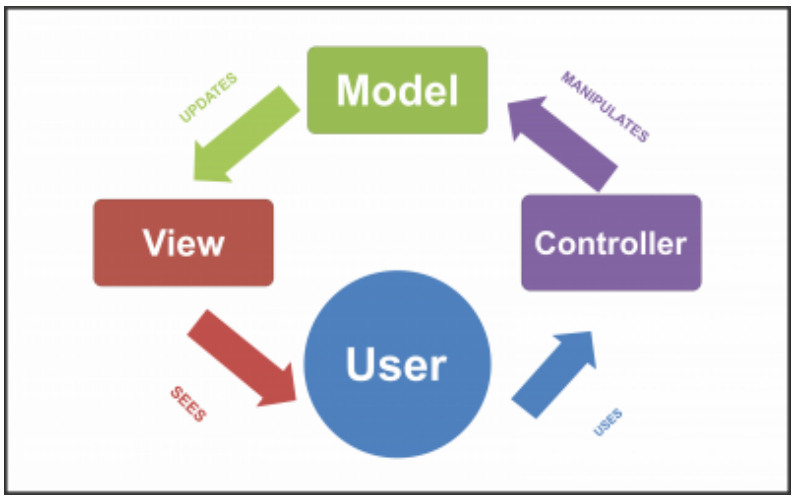


Figura 3.1: Schema dell'architettura Model-View-Controller.

In Django, come si può vedere in Figura 3.2, avviene qualche cambiamento rispetto a questa architettura, poiché il Controller diventa il framework stesso, entra in gioco il Template che prende il posto della View, mentre quest'ultima, che prima decideva come mostrare i dati, ora si limita a determinare quali dati mostrare. In particolare:

- il Model: definisce i modelli del sito (le entità), che vengono poi trasformate in tabelle a database;
- il Template: definisce come visualizzare i dati sulla pagina web, svolgendo un ruolo simile alla View dell'architettura MVC[13];
- la View: gestisce richieste e risposte HTTP; essa ottiene un oggetto HTTPRequest come paramentro (generalmente denominato request) [14] e restituisce un oggetto HTTPResponse restituendolo tramite redirect (per indirizzare ad un nuovo URL)[15] o render (se non cambia l'URL della pagina visualizzata).

I modelli sono classi Python, costruite su misura del progetto in questione nel file models.py, e contengono attributi che corrispondono alle colonne delle rispettive tabelle a database, mentre ogni riga di quelle tabelle è una specifica istanza[10]. La vista, successivamente si occupa, una volta ricevuta una richiesta, di eseguire una query a database che recuperi le istanze desiderate per eseguirne un rendering e

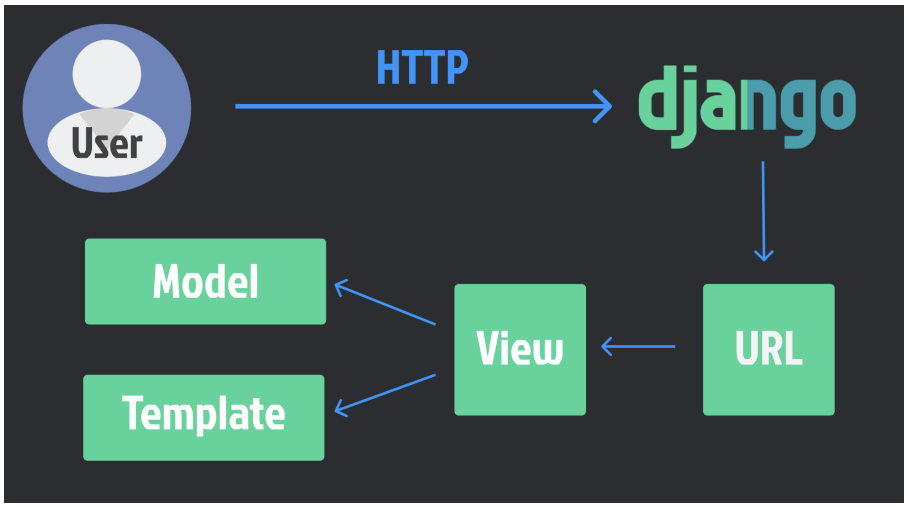


Figura 3.2: Schema dell'architettura Model-Template-View.

restituire una risposta HTTP. Le viste vengono richiamate tramite il file urls.py (e gli url) per poi reindirizzare i template html[16]. Infine il template fornisce il layout che viene visualizzato dall'utente finale: la loro componente più "dinamica" è principalmente merito dei tag django, che fungono un po' come un ponte tra la vista e il template, permettendo a quest ultimo di visualizzare (o meglio, far visualizzare), i dati desiderati e restituiti dalle views. Questi tag usano un linguaggio molto simile a Python, ma che in realtà è tipico di Django, e che permette ad esempio di ciclare su una serie di dati ricevuti o di eseguire condizioni di verifica (proprio come l'if di Python). Mentre i tag di Django sono racchiusi tra parentesi graffe e percentuali, le variabili di Django sono invece racchiuse tra doppie graffe e rappresentano esattamente i dati da visualizzare, presi dal modello. I template possono inoltre essere estesi, così come è stato fatto in questo progetto, definendo una struttura di base (in "base.html") e poi facendola ereditare da tutti gli altri file con estensione .html: un esempio di come funziona e come può essere utile è dato proprio dal menù (navbar) di questo progetto, che è stata posta nel template base affinché non dovesse essere ripetuta per ogni singola pagina, e allo stesso tempo è stata "nascosta" per le due pagine in cui non serviva, ovvero la pagina di login e quella di registrazione. Già da questa breve panoramica del framework Django emerge quanto esso sia focalizzato sul risparmio di tempo e righe di codice, puntando a scrivere solo lo stretto necessario ed a riutilizzare tutto quello che è possibile.

3.1.2 ORM

L'Object-Relational Mapping (ORM) è uno dei punti chiave che rendono Django uno strumento così potente e utilizzato. Come mostrato in Figura 3.3, esso funge da intermediario tra il linguaggio Python ed il database [13] creando oggetti Python appunto, per interagire, creare e modificare questi senza il bisogno di scrivere query SQL ad ogni richiesta necessaria. Già da questa breve panoramica si può capire lo stretto legame quindi tra l'ORM con i modelli ed i queryset; i primi che definiscono la struttura degli oggetti, ed i secondi usati come strumento più rapido ed efficace per interrogare il database.

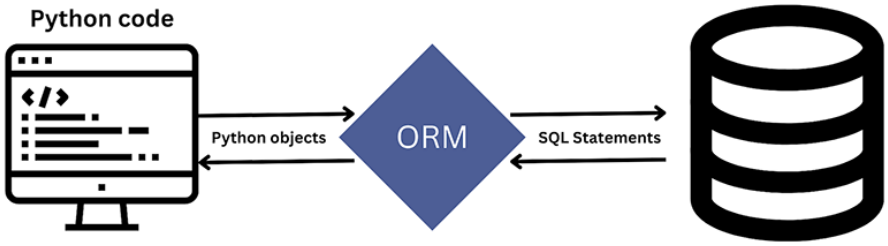


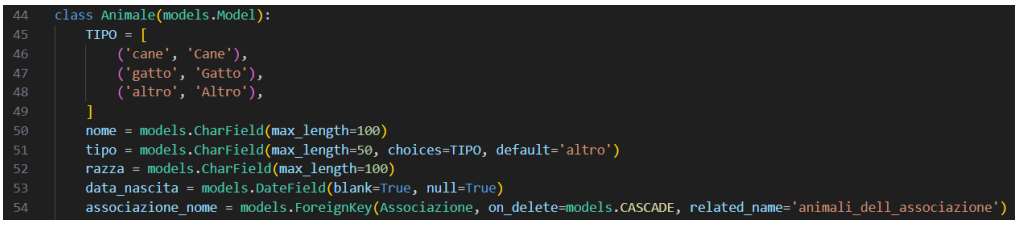
Figura 3.3: Funzione dell'ORM di Django.

L'ORM permette di eseguire facilmente operazioni CRUD (Create, Read, Update, Delete) sui modelli, ovvero di creare, leggere, aggiornare (modificare) o cancellare i record già esistenti a database, sempre ovviamente usando solo codice Python. Questo non solo facilita e velocizza la scrittura di codice, ma lo rende anche più portabile e manutenibile, cercando di eliminare quanto più possibile il legame con un tipo specifico di database, così che Django possa operare in un panorama più vasto possibile (e favorendo di molto la sua distribuzione e popolarità).

Attraverso l'Object-Relational Mapping è possibile anche gestire le relazioni tra modelli, cioè legami tra una "classe" ed un'altra che definisce una connessione logica. Le relazioni sono di tre tipi:

- ForeignKey
- ManyToManyField
- OneToOneField

La Foreign Key è una relazione "uno a molti" in cui un oggetto del modello sorgente può avere più relazioni con il modello destinazione, ma non vale il contrario: nel progetto in questione un esempio di ciò è dato dal modello "Animale" e dall'attributo "associazione_nome" legato a questo tramite una ForeignKey e visibile in Figura 3.4. In questo caso infatti un animale può essere legato ad una sola associazione di appartenenza, ma al contrario la stessa associazione può avere più animali in custodia. Diversamente la Many To Many Field rappresenta una relazione molti a molti, che nel caso precedente corrisponderebbe (a livello unicamente teorico) ad associazioni che hanno diversi animali, ma in cui anche questi ultimi possono appartenere contemporaneamente a diverse associazioni. Questo tipo di relazione si presta bene ai casi di social network in cui utenti uguali tra loro possono seguirsi a vicenda, mentre nel progetto di questa tesi trova poco spazio.



```
44  class Animale(models.Model):
45      TIPO = [
46          ('cane', 'Cane'),
47          ('gatto', 'Gatto'),
48          ('altro', 'Altro'),
49      ]
50      nome = models.CharField(max_length=100)
51      tipo = models.CharField(max_length=50, choices=TIPO, default='altro')
52      razza = models.CharField(max_length=100)
53      data_nascita = models.DateField(blank=True, null=True)
54      associazione_nome = models.ForeignKey(Associazione, on_delete=models.CASCADE, related_name='animali_dell_associazione')
```

Figura 3.4: Blocco di codice relativo ad una parte del modello Animale nel file muovimodelli/models.py.

Infine la One To One Field corrisponde ad una relazione "uno a uno" esclusiva, quindi un unico modello sorgente ha un'unica relazione con il modello destinazione e viceversa [13]. In questo caso ci sono due esempi significativi in questo progetto, poiché questa relazione, come mostrato in Figura 3.5 viene impiegata per legare un proprietario ad un'associazione nel modello "Associazione", poiché quindi un'associazione può avere un solo proprietario e viceversa, ma viene anche usata per un'altra funzione molto utile di Django, ovvero estendere il modello user. Django infatti mette a disposizione un modello "user" con i più comuni attributi (come nome e cognome o password), sempre con lo scopo di facilitare il lavoro ai programmatori e scrivere meno righe di codice possibile; per prelevare dall'utente tutte le informazioni ritenute rilevanti a questo caso quindi, si è creato un modello Utente che "estendesse" il modello user, e lo si è fatto inserendo quest ultimo come attributo dell'Utente, attraverso un OneToOneField, poiché infatti ogni utente corrisponderà ad un solo user e viceversa. L'user di Django non serve solo in fase di creazione dei modelli, ma ha anche una quantità di altre caratteristiche che lo rendono molto utile, come il login e l'autenticazione già implementata in Django.

Per avere un quadro completo di come funzionino modelli e database all'interno di tutto questo processo si devono infine citare le migrazioni. Le migrazioni sono

```
25  class Associazione(models.Model):
26      proprietario = models.OneToOneField(User, on_delete=models.CASCADE, related_name='associazione')
27      nome = models.CharField(max_length=100)
28      codice_fiscale = models.CharField(max_length=16, unique=True)
```

Figura 3.5: Blocco di codice relativo a una parte del modello Associazione del file nuovimodelli/models.py.

uno strumento che automatizza il passaggio da modelli Python a tabelle su database in due passaggi: il primo, che avviene tramite il comando "makemigrations" da riga di comando, che permette a Django di interpretare le modifiche apportate ai modelli e convertirle in istruzioni da dare a database con un linguaggio appropriato, il tutto tramite la creazione di un file di migrazioni che contenga quanto appena spiegato; il secondo passaggio, tramite il comando "migrate" sempre da riga di comando, applica effettivamente a database le modifiche precedentemente viste seguendo le istruzioni del file sopra citato. Tutto questo processo permette sia di tornare eventualmente a migrazioni (e quindi stati del database) precedenti attraverso il processo di "rollback", ma soprattutto di preservare la coerenza dei dati e la loro integrità man mano che il progetto evolve e si modifica, mantenendoli sincronizzati allo stato attuale[13].

3.1.3 Alternative a Django

Tra i "competitor" di Django vi è sicuramente Flask, un micro-framework sviluppato con lo scopo di fornire una struttura leggera ma potente, e di non essere intrusivo, ovvero senza imporre una struttura specifica e predefinita ma lasciando il più possibile liberi gli sviluppatori di scegliere le proprie librerie e gli strumenti da utilizzare [17]. Si tratta in entrambi i casi di framework Python ed entrambe valide scelte, ma si differenziano enormemente per il tipo di progetto da sviluppare, e soprattutto per la sua complessità e per le dimensioni. In particolare, mentre Flask è meno complesso ed ha una curva di apprendimento meno ripida rispetto a Django, risultando quindi più alla portata di tutti, è ottimo però per progetti di piccole dimensioni ed offre solo il minimo indispensabile per progettare un'applicazione. Inoltre un grande vantaggio di Django consiste nel suo ORM integrato, che Flask non possiede, e infatti in questo caso è lo sviluppatore a dover gestire interamente l'interazione con il database ed eseguire le query necessarie. Similmente, Flask non possiede neanche meccanismi di sicurezza incorporati, lasciando anche questo carico interamente sulle spalle del programmatore, e costituendo quindi uno svantaggio non solo in termini di tempo e carico di lavoro, ma anche di sicurezza stessa. I principali punti di forza del micro-framework consistono nell'essere leggero e flessibile, ed infatti non possiede tante strutture che invece è in grado di fornire Django proprio per restare il meno intrusivo possibile, ma questo comporta, come

nei casi sopra citati e anche in molti altri, più complessità per lo sviluppatore ed un maggiore carico di lavoro per implementare funzioni ad ogni livello del progetto: un esempio paradigmatico è fornito dall'interfaccia amministrativa preconfigurata di Django, suo punto di forza per gestire più facilmente l'amministrazione del proprio sito, che in Flask non esiste ed in cui, per ottenere lo stesso risultato, serve rivolgersi ad estensioni di terze parti. Entrambi sono considerati scalabili, ma Django viene comunemente considerato la prima scelta per siti o applicazioni che prevedono un grande traffico, mentre Flask è molto adatto a piccoli siti o alla creazione rapida di prototipi. In generale la scelta si consiglia che ricada su Django quando il progetto da sviluppare possiede le seguenti caratteristiche: si tratta di un'applicazione complessa, è fondamentale l'aspetto della sicurezza, serve un ORM potente ed un sistema di amministrazione.

Un'altra alternativa nel panorama dei framework è Ruby on Rails (RoR), che, come Django, è un framework open-source ed insieme sono due dei più popolari al mondo[18]. La prima differenza però risiede già nei linguaggi di programmazione su cui si basano, infatti Django è scritto in Python, e risulta quindi più scalabile di Rails, che si basa su Ruby[19]. Un altro importante confronto riguarda i database supportati: Django infatti è compatibile con svariati database (come MySQL, PostgreSQL o Oracle) mentre RoR supporta solo SQLite. Inoltre quest'ultimo ha un'architettura MVC (Model-View-Controller), che come visto precedentemente differisce dall'architettura MVT di Django. Infine Django è una scelta nettamente migliore per quanto riguarda la sicurezza, per tutte le già citate componenti integrate che fornisce in questo campo, e per la scalabilità, che lo rendono più adatto a gestire grandi quantità di traffico ed intensi carichi di dati.

Un'altra valida scelta è data da Laravel, un framework web gratuito ed open source che si basa su PHP. Esso ha un'architettura MVC e si basa su un altro framework PHP, cioè Symfony[20]. Laravel ha una sintassi elegante ed un packaging modulare; possiede un Eloquent ORM che permette anche di effettuare query a database sfruttando la sintassi PHP. Ponendo i due framweork a confronto, non emergono enormi differenze in termini di complessità, che al massimo può riguardare la facilità soggettiva di conoscere (o imparare per chi non lo conoscesse) Python piuttosto che PHP, il ché potrebbe rendere Django leggermente più semplice. Sempre il linguaggio utilizzato però crea delle importanti differenze a livello di prestazioni, così che l'utilizzo di Python renda Django più veloce e performante ed aiuta anche a risolvere problemi nel codice in modo rapido; dall'altra parte Laravel è molto solido, ma tutte le funzionalità che offre possono penalizzarlo in termini di velocità. Per quanto riguarda l'architettura, la MVT facilita una creazione dinamica e maggiormente personalizzata, rendendo Django adatto a contesti che necessitano di molteplici, continue e rapide modifiche e supportano anche grandi dimensioni. A tal proposito, anche grazie al linguaggio su cui si basa, Django è nettamente più

scalabile di Laravel, e può lavorare con diverse tecnologie mantenendo una grande flessibilità anche nel rispondere a nuove richieste o nuove tendenze di mercato. Sempre a livello di linguaggi, Python offre una sicurezza nettamente maggiore di PHP, rendendo Django, anche grazie a questo, una scelta ovvia tra i due nei casi in cui la sicurezza sia una parte fondamentale del progetto. Per quanto riguarda invece le librerie ed i database supportati non si riscontrano grandi differenze tra i due.

Il confronto tra questi framework è sinteticamente riportato in Tabella 3.1, per mettere in luce i pro e contro di ognuno di essi e le principali differenze, soprattutto rispetto a Django. In conclusione infatti, la scelta di questo progetto è inevitabilmente ricaduta su Django per diversi motivi: intanto per valorizzare la sicurezza ed implementare i controlli necessari in questo ambito con facilità e rapidità; in secondo luogo la speranza è ovviamente di vedere prendere forma nel concreto a questo progetto e che possa coinvolgere il maggior numero possibile di utenti e associazioni, quindi si è voluto predisporlo ad un possibile futuro intenso traffico, avvalendosi della forte scalabilità del framework. Ovviamente anche la facilità e la rapidità che Django offre nell'implementare funzioni di base (che riguardino la sicurezza, l'amministrazione o altro) sono punti a favore nella sua scelta.

Framework	Pro	Contro	Differenze principali
Django	Framework completo a "batteries included"; ORM integrato e potente; elevata sicurezza già incorporata; scalabile e adatto a grandi progetti; interfaccia admin automatica.	Ripida curva di apprendimento; elevata complessità iniziale; meno flessibile per micro-progetti.	Più complesso ma più completo di Flask; più scalabile e sicuro di RoR e Laravel.
Flask	Leggero e flessibile; rapida curva di apprendimento; ideale per piccoli progetti.	Mancanza di ORM integrato; nessun sistema di sicurezza incorporato; maggior lavoro manuale per funzioni comuni.	È un micro-framework contro un framework completo; meno adatto a progetti complessi o con alto traffico.
Ruby on Rails (RoR)	Framework maturo e popolare; architettura MVC chiara.	Minore scalabilità di Django; supporta solo SQLite di default; basatu su Ruby (meno performante).	Architettura MVC vs MVT; meno scalabile e sicuro di Django.
Laravel	Sintassi elegante e modulare; buon ORM; comunità attiva.	Basato su PHP quindi meno sicuro e performante; più lento.	Architettura MVC vs MVT; meno veloce, sicuro e scalabile di Django.

Tabella 3.1: Confronto tra Django ed i framework concorrenti.

3.2 Docker

Docker è un software opern source che consente di creare, aggiornare, implementare, eseguire e gestire container[21]. I container sono l'elemento fondamentale di questa piattaforma: essi combinano il codice dell'applicazione con le dipendenze del sistema operativo in modo facile, veloce ed efficiente. In sostanza permettono di eseguire più componenti di un'applicazione, appoggiandosi ad un unico sistema operativo host, in modo molto simile a come un hypervisor permette a più macchine virtuali di condividere e utilizzare le stesse risorse di un unico server. Docker infatti offre gli stessi vantaggi delle virtual machine: isolamento, scalabilità, leggerezza, produttività ed efficienza.

I due concetti fondamentali per capire Docker sono: immagine e container. L'immagine contiene tutto ciò che è essenziale ad eseguire il software, come il codice, gli strumenti, le librerie e molto altro. Un container invece è un'istanza in esecuzione di un'immagine Docker [22], di fatto è l'applicazione creata dall'immagine, che poi realmente va in esecuzione, mentre l'immagine predispone per essa tutto ciò che è necessario alla sua realizzazione. Banalizzando un po' il concetto funziona quasi come una torta creata a partire da una ricetta, e proprio così a partire dalla stessa immagine si possono creare più container. Questi ultimi, condividendo il kernel del sistema operativo, sono più leggeri da mantenere e possono essere avviati molto velocemente.

3.2.1 Componenti principali

Il principale componente di un progetto che sfrutta Docker è il Dockerfile. Si tratta di un file di testo contenente una serie di "istruzioni" che indicano al software come creare un'immagine. Ogni sua istruzione realizza uno strato dell'immagine finale, che ad esempio può essere un semplice copiare un file o una cartella o creare variabili d'ambiente. Per costruire un'immagine da un dockerfile è necessario un solo comando, "docker build", mentre per eseguire un nuovo container il comando è "docker run". Un'istruzione fondamentale del dockerfile è costituita dalla variabile d'ambiente ENV, che permette di impostare chiavi e valori di default per configurare i container [23].

Un'istruzione altrettanto importante è sicuramente costituita da EXPOSE. I container sono per loro natura "chiusi", e nessuno può averne accesso dall'esterno. Questa istruzione permette di esporre la porta agli altri container dello stesso host, ma comunque non al mondo esterno, per cui sarebbe necessaria la pubblicazione di quella porta[23]. Quindi questo è sostanzialmente il comando che permette di far

girare il programma in locale e poter vedere, una volta avviato il container ed effettuato l'accesso alla porta impostata, le varie modifiche del sito o dell'applicazione in questione in tempo reale.

Come già riportato, i due principali elementi di questo software sono i container e le immagini. I container però, sono creati per essere stateless e indipendenti: tutto ciò che riguarda un container rimane dentro ad esso, e così anche i dati. Se infatti un container venisse eliminato verrebbe eliminato di conseguenza tutto ciò che contiene. Per evitare tutto ciò ci si serve di un altro elemento rilevante, i volumi. Essi sono infatti archivi dati persistenti per i container [24]. Si tratta di uno speciale spazio in cui i container possono salvare i propri dati per evitare che vadano persi anche se venisse cancellato il container. Essi quindi persistono anche al di fuori del ciclo di vita di un container. Inoltre più container possono scrivere nello stesso volume ed un container può essere aggiornato o sostituito senza la perdita di dati importanti.

La parte più importante, però, è il Docker Engine, la sua architettura, rappresentata in Figura 3.6. Il Docker Engine è il software che permette la creazione, l'avvio e la gestione dei container, sostanzialmente il cuore del funzionamento di Docker. Esso è costituito da tre componenti:

- Docker Daemon
- Docker CLI
- Docker REST API

Il Daemon di Docker crea e gestisce immagini utilizzando i comandi del client [21]. Si tratta di un processo in background che resta in ascolto di richieste API e gestisce gli oggetti di Docker, come immagini, container e volumi [24]. Docker CLI (client) è l'interfaccia a riga di comando che permette agli utenti di interagire, tramite le API Docker, con il Daemon. La REST API è il modo tramite il quale queste due componenti comunicano tra loro, e si tratta quindi di un'interfaccia che permette di interagire con il Daemon.

Un esempio del loro funzionamento può essere dato dal comando "docker run immagine_desiderata". Una volta lanciato questo comando da parte dell'utente, e quindi attraverso l'API REST, il Docker CLI la manda al Deamon che: controlla se l'immagine è presente e in caso contrario la scarica, crea un nuovo contenitore basato su quell'immagine assegnandoli risorse dedicate come CPU e spazio di archiviazione, configura la rete ed avvia il container.

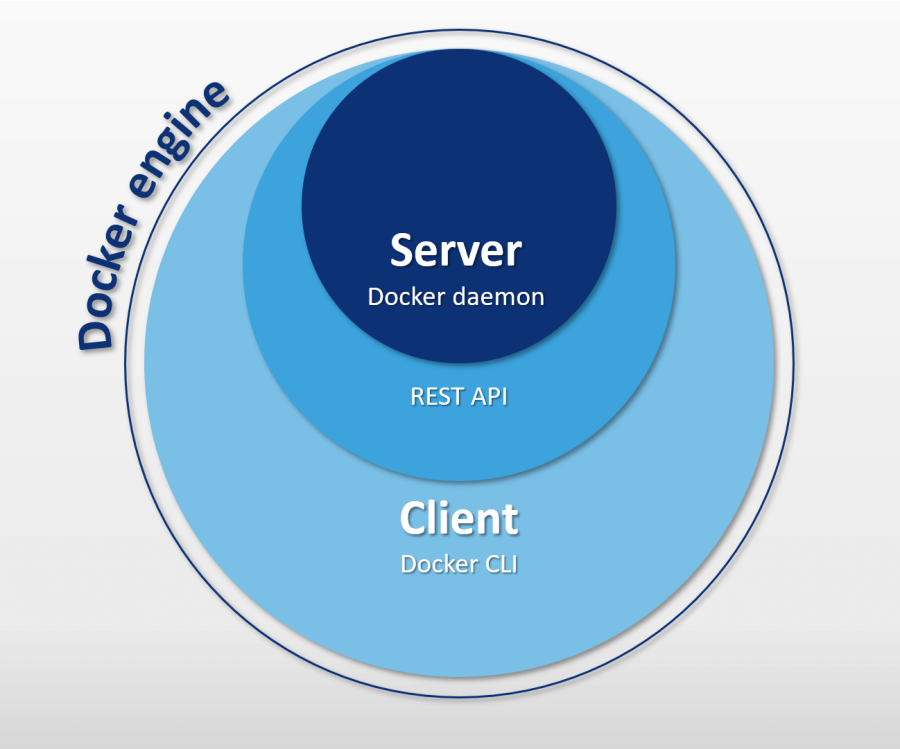


Figura 3.6: Architettura del Docker Engine.

3.2.2 Docker vs Macchine Virtuali

Le macchine virtuali (VM) sono emulazioni software di piattaforme di computer fisici[25], chiamati host, ovvero simulano un vero e proprio computer all'interno di un altro computer fisico. Ogni VM, come mostrato in Figura 3.7, ha un proprio sistema operativo, eseguito su un software che è l'hypervisor. Quest'ultimo si occupa di virtualizzare le risorse hardware e dividerle tra le varie macchine virtuali. Tutto ciò comporta un grande isolamento delle macchine, che permette ad una di continuare a funzionare a prescindere che altre si siano bloccate, ma richiede una grande quantità di risorse, comportando quindi tempi di avvio lunghi ed un notevole consumo di memoria e CPU.

Dall'altra parte, Docker, funziona in modo molto diverso: non simulano interi computer, ma creano ambienti isolati, che condividono alcune risorse tra loro, come il kernel del sistema operativo dell'host. Quindi non dovendo gestire diversi sistemi operativi e troppe risorse dedicate, Docker risulta molto più leggero e veloce delle macchine virtuali e fa sì che i container possano essere spostati tra sistemi diversi poiché contengono al loro interno già tutto quello che è necessario al loro funzionamento.

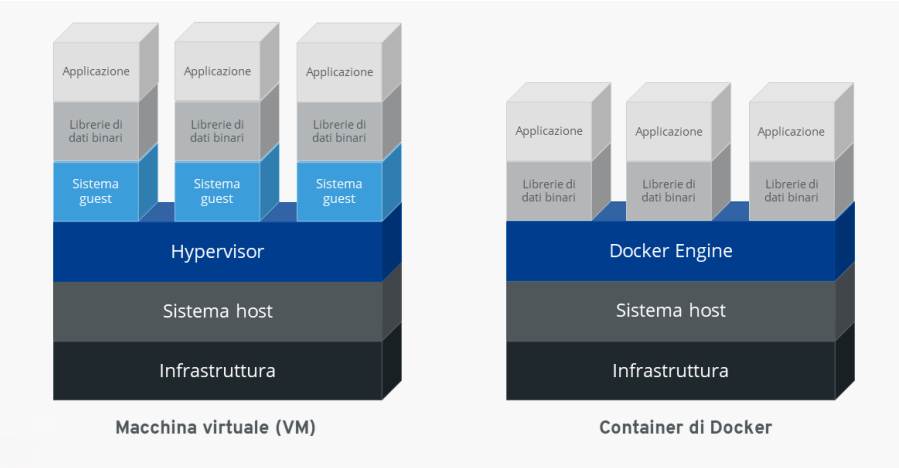


Figura 3.7: Confronto tra macchine virtuali e Docker.

La prima differenza quindi risiede nell'architettura[26] delle due, che non costituisce un vantaggio per l'una o per l'altra ma determina i contesti più adatti al loro impiego. Ad esempio se è necessario avere più sistemi operativi sarà consigliato usare una macchina virtuale, mentre se non si hanno sufficienti risorse da dividere è meglio Docker.

Per quanto riguarda l'efficienza, invece, poiché i container condividono lo stesso kernel del sistema host, risulteranno più efficienti per quanto riguarda il consumo di risorse, e questo li rende anche più veloci da distribuire. Anche per le prestazioni risulta migliore Docker, grazie alla rapidità di avvio dei container, che al contrario delle macchine virtuali non rischiano di avere latenze a causa dell'avvio di un nuovo sistema operativo. In ambito di isolamento invece entrambe forniscono ambienti isolati, ma differiscono per il livello di isolamento: quello di Docker è un isolamento a livello di processo, mentre le VM hanno un isolamento completo, quindi maggiore e di conseguenza ottengono un vantaggio in termini di sicurezza, poiché una compromissione all'interno di una macchina virtuale non rischia di avere conseguenze sulle altre. Per quanto concerne la portabilità invece, le macchine virtuali, nonostante possano essere migrate tra vari ambienti, a causa delle loro dimensioni e della configurazione di cui necessitano risultano più difficili da spostare rispetto ai contenitori docker, che racchiudono in un'unica unità tutto ciò che è necessario al loro avvio, e pertanto possono essere spostati in qualsiasi ambiente contenga Docker. Sempre per la loro architettura, Docker risulta anche più facilmente gestibile e mantenibile, soprattutto su larga scala.

3.2.3 Vantaggi di Docker

Docker è lo strumento di containerizzazione più utilizzato, e alla base della sua popolarità ci sono un importante numero di vantaggi:

- Innanzitutto si tratta di una piattaforma open-source[27], quindi gratuita e fruibile a tutti, ma anche con molte informazioni reperibili facilmente online, che rendono "semplice" iniziare ad utilizzarlo, o meglio fanno sì che l'unica condizione necessaria sia voler imparare a padroneggiarlo.
- I container consentono di svolgere lavori maggiori delle capacità hardware presenti [27], agevolando di molto gli sviluppatori ed abbattendo i costi del ridimensionamento lì dove il sito inizia a svilupparsi abbastanza da necessitare di nuove risorse hardware.
- Consente una migliore distribuzione del software poiché, basandosi completamente su dei container, che sono portatili e autonomi e che contengono tutte le dipendenze necessarie, vengono eliminati i problemi di variabilità della configurazione ed il container eseguito produrrà lo stesso risultato a prescindere dal luogo fisico in cui viene avviato.
- Inoltre i container rendono le applicazioni più flessibili e resilienti.
- L'interfaccia consente di definire reti isolate per i container, apportando un grande guadagno in termini di sicurezza.
- Infine sono molto efficienti nell'ambito dei microservizi, ovvero singole funzioni con un unico obiettivo. I microservizi sostanzilamente "fanno una sola cosa e la fanno bene"[27], e Docker con i suoi container aiuta a renderli autonomi, flessibili ed implementabili.

3.3 Database Relazionali

Un database è un insieme di dati o informazioni strutturate, e generalmente archiviate in un sistema informatico [28]. Il termine database (DB) indica generalmente un archivio di dati, strutturati con collegamenti logici che ne favoriscano la gestione e soddisfino le richieste degli utenti che si interfacciano a loro tramite i "query language" [29]. I database possono essere di diverso tipo.

I più diffusi, almeno fino a qualche anno fa, erano i database relazionali, in cui i dati sono organizzati in tabelle strutturate composte da righe e colonne [28]. Ad

opporsi a questi vi sono i database non relazionali, ovvro che non seguono il modello relazionale di tabelle con rigide relazioni riga-colonna. Tra i database non relazionali quindi si trovano: database orientati agli oggetti, in cui le informazioni sono appunto rappresentate da oggetti; database gerarchici che archiviano le informazioni secondo strutture ad albero; database grafici; database NoSQL, spesso utilizzati come sinonimo generico dei db non relazionali, che permettono di gestire dati non strutturati e semi-strutturati [28].

3.3.1 La teoria di Codd

Il modello relazionale fu proposto da Edgar F. Codd nel 1970 nel suo articolo "A Relational Model of Data for Large Shared Data Banks"[30], e per questo viene considerato il padre del database relazionale. In quell'articolo Codd cercava di unire il concetto di "relazione" con quello di "tabella". Lo scopo era quello di eliminare complicati file gerarchici per lasciare il posto a delle più intuitive tabelle, ognuna delle quali doveva rappresentare un tipo di informazione. Le tabelle (Figura 3.8) hanno:

- delle righe, o tuple: rappresentano gli effettivi elementi che "popolano" quella tabella;
- delle colonne, chiamate attributi: che rappresentano le varie "caratteristiche" di quell'elemento.

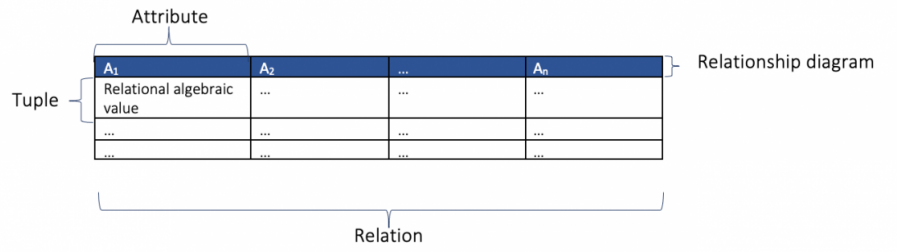


Figura 3.8: Esempio di una tabella di un database relazionale.

Ogni attributo possiede un dominio, ovvero un range di valori che può assumere. Codd inoltre introduce il concetto di chiave primaria: ciò che permette di identificare un elemento (tupla) e di conseguenza di rendelo "rintracciabile" dalle altre tabelle, sotto forma di chiave esterna (ovvero la chiave primaria di una tabella al di fuori di quella in esame). In sostanza il modello relazionale si basa sulla teoria degli insiemi, e su come si possono "combinare" insiemi di oggetti. Ed esattamente come

la matematica lavora sugli insiemi Codd propose delle operazioni matematiche che similmente lavorassero sulle tabelle estrapolando dati o creando nuove tabelle da quelle di partenza, e tutto ciò condusse poi allo sviluppo del linguaggio SQL[31].

In seguito Codd propose 12 regole (13 se si considera anche la numero zero) per identificare un DBMS (Database Management System) davvero relazionale. Si tratta di regole molto rigide, e anche costose, per cui nessuno riesce a soddisfarle tutte, ma servono sicuramente come guida per orientarsi e continuare a migliorarsi. La regola più importante è sicuramente la regola zero, o regola base, che indica che un sistema di gestione di basi di dati relazionali, per essere ritenuto tale, deve essere in grado di gestire i dati unicamente attraverso le sue capacità relazionali[32]: questo indica che tutti i dati, metadati e operazioni debbano essere gestite tramite un modello relazionale, e che le manipolazioni debbano avvenire tramite un linguaggio relazionale (come SQL).

Con la regola uno invece Codd esplicita come tutte le informazioni debbano essere riportate in tabella, questo indica non solo i dati principali ma anche i metadati ad esempio, o eventuali autorizzazioni e chiavi: tutto deve essere chiaro e strutturato con la medesima logica, garantendo così uniformità e rendendo il database completo, poiché non necessita di file esterni per essere compreso. La regola numero due garantisce che ogni valore presente sia univocamente identificabile, esplicitando come per ottenere un preciso dato, siano necessari e sufficienti solo tre informazioni: il nome della tabella, il valore di chiave primaria ed il nome della colonna; questo permette di eliminare ambiguità o duplicati, verificando che non esistano due valori appartenenti alla stessa tabella e riferiti al medesimo elemento (dato dalla chiave) per l'attributo richiesto. Con la regola tre vengono date precise istruzioni su come gestire i valori mancanti o non accettabili attraverso il valore NULL, che non corrisponde a zero, ma è una sorta di segnaposto: questo permette sia di rendere l'archivio di dati più realistico in un mondo in cui regna l'incertezza e non si possono sempre ottenere tutte le informazioni che si desiderano, sia più coerente dal punto di vista logico, poiché non corrisponde a nessun altro valore e i dati mancanti (e quindi con null) vengono gestiti tutti allo stesso modo.

Con la regola numero quattro, Codd mira a rendere il sistema auto-descrittivo e completo, indicando come tutte le informazioni, anche quelle relative alla struttura stessa del database, debbano essere registrate in tabelle e conservate con le stesse procedure degli altri dati. Con la quinta regola invece mira a definire un linguaggio standard e unico per la manipolazione dei dati, ed infatti da qui poi si sviluppa il linguaggio SQL, eliminando così la dipendenza da strumenti o linguaggi esterni e rendendo il sistema portabile e standard. Con la regola successiva, Codd sostiene che una vista che sia teoricamente aggiornabile, allora deve poterlo essere anche dal sistema, permettendo in questo modo di lavorare e aggiornare più velocemente i dati senza doversi preoccupare troppo del database. Con la settima regola si ha

un'evoluzione dal modello procedurale che era solito usare e che necessitava una modifica elemento per elemento, per arrivare a poter eseguire operazioni su intere tabelle o set di dati, quindi ponendo ora le operazioni ad un "alto-livello".

Successivamente Codd rende gli utilizzatori del database "indifferenti" al livello fisico e alle informazioni che lo riguardano, determinando che le modifiche di questo aspetto non debbano intaccare il database e le sue tabelle. Similmente con la numero nove si rende il sistema il più possibile indipendente dallo schema logico, ovvero delle modifiche, anche alle tabelle stesse, non devono interrompere il suo corretto funzionamento. La regola dieci indica che l'integrità sia gestita all'interno del database stesso, garantendo così coerenza e correttezza dei dati e faverendone la mantenibilità. Successivamente Codd si preoccupa anche della complessità percepita dagli utilizzatori, indicando come questa debba essere invisibile all'occhio esterno, e anche i dati si trovassero su più server, chi ne ha l'accesso deve percepire che provengano tutti dallo stesso luogo. Infine, non deve essere possibile utilizzare linguaggi di basso livello (che ad esempio modificano riga per riga) per aggirare o sovvertire le regole relazionali espresse con un linguaggio superiore, cioè sostanzialmente si devono utilizzare regole e linguaggi relazionali, garantendo così sicurezza e integrità.

3.3.2 Relazionale vs NoSQL

I database relazionali (tra cui quelli SQL) sfruttano un linguaggio strutturato per accedere e manipolare i dati archiviati, ed offrono le informazioni in rigide tabelle, in cui lo schema costituisce il nome (l'etichetta che indica ciò che viene rappresentato), mentre le varie righe, anche dette istanze, sono le informazioni memorizzate.

I database NoSQL invece raggruppano le informazioni in delle collezioni; queste contengono insiemi di documenti, ognuno dei quali è costituito da un elenco di coppie "chiave"-"valore" e questi campi possono contenere qualsiasi tipo di dato, senza avere uno schema di base o dover definire prima la struttura dei documenti, e anche senza la necessità che i documenti contengano gli stessi campi, come rappresentato in Figura 3.9.

La prima differenza che emerge si trova nella primissima fase di creazione di un database e riguarda la struttura stessa, che da un lato, quello dei db relazionali, deve essere rigida e determinata anticipatamente, creando appositi modelli ad esempio con tutti i campi (attributi) necessari alle "entità" che verranno rappresentate. Dall'altro lato, i db NoSQL non necessitano proprio di una struttura, che quindi non richiederà del tempo per essere definita rendendo il processo sotto alcuni puti di vista più rapido, per lo meno in fase di immagazzinamento dei dati, e permettendo

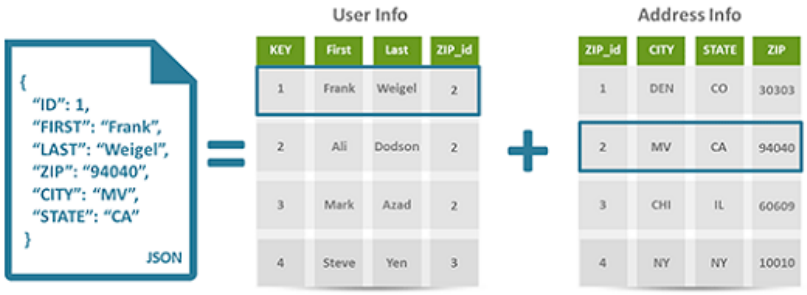


Figura 3.9: Database NoSQL e relazionali a confronto.

quindi di "modificare" questo modello, che non è definito, per ogni "oggetto" e in base alle esigenze specifiche[28]. Data questa prima definizione e differenziazione, si può facilmente capire il perché del rapido sviluppo che hanno ottenuto questi ultimi database, che permettono una memorizzazione di dati molto più varia, rapida e favoriscono la flessibilità.

I database relazionali sono incentrati sulle proprietà ACID[28]. Essi infatti garantiscono:

- Atomicità: le transazioni vengono considerate come un blocco unico, quindi o viene eseguita per intero o non viene proprio effettuata nemmeno in parte;
- Consistenza: garantisce la presenza di tutte le condizioni necessarie e non permette duplicazioni inutili;
- Isolamento: tratta ogni transazione come unica e non permette di vederne gli effetti fino al completamento di essa, così che non si creino conflitti con altre transazioni ed in modo tale da non creare confusione;
- Durabilità: assicura la durata nel tempo delle varie modifiche una volta portata a termine la transazione.

I database non relazionali sacrificano alcune di queste proprietà, in particolare la consistenza ad esempio, in favore di maggiore flessibilità e scalabilità.

Così come i db relazionali offrono molti vantaggi, hanno però anche degli svantaggi. Innanzitutto vi è una potenziale complessità nel gestire ed effettuare le query; inoltre, a seconda della complessità della richiesta o alla vastità dei dati presenti, possono esserci perdite nelle prestazioni[33]. La sicurezza poi, tematica molto importante in questi ambiti, può riportare alcuni problemi se non viene trattata nel modo corretto e completo. Vi sono infatti attacchi, come l'SQL injection, che possono compromettere l'intero database: nel caso in questione si tratta di introdurre

query secondo linguaggio SQL in campi di immissione come dei form, per recuperare, manipolare o distruggere dati sensibili[34]. Infine, i sistemi per gestire database relazionali, possono comportare importanti costi di gestione e manutenzione.

Dall'altra parte, i database non relazionali offrono diversi vantaggi, soprattutto se si lavora con una grande quantità di dati in rapido cambiamento. Innanzitutto essi godono di un'importante scalabilità orizzontale: a differenza di quelli relazionali che, per la loro scalabilità verticale, quando i dati da gestire diventano troppi, viene potenziato il singolo server, i db non relazionali possono godere di più server e quindi rispondere al maggiore carico semplicemente aggiungendo un nuovo server, e questo è ottimo per situazioni in cui non si conosce a priori il traffico ed il carico che ci sarà[33]. Inoltre, la flessibilità intrinseca ad un database privo di una struttura definita, rende questa scelta vantaggiosa per situazioni in cui i dati possono cambiare nel tempo e possono includere una grande varietà di campi dei record da un documento all'altro. Essi offrono anche prestazioni elevate ed un'ottima gestione del carico, poiché numerosi db implementano meccanismi come lo sharding automatico che divide il carico di lavoro su più server, o il bilanciamento che permette di suddividere le richieste in arrivo su più server per evitare il sovraccarico. Infine sono ottimi nella gestione di dati non strutturati o semi-strutturati, come i commenti degli utenti o documenti JSON, e poiché i dati in arrivo dagli utenti sono spesso imprevedibili e possono assumere svariate forme sono ad oggi una scelta molto popolare in alcuni contesti.

Dall'altro lato della medaglia, la mancata standardizzazione dei database non relazionali e la vastità di forme in cui si possono sviluppare può complicare il loro impiego e anche solo la scelta più efficiente al caso, senza contare le competenze specialistiche che può necessitare. Inoltre, come già accennato, viene spesso sacrificata la consistenza dei dati in favore della flessibilità e della disponibilità, e questo è sicuramente un problema rilevante, specialmente in contesti, come banche o ospedali, in cui l'accuratezza dei dati è fondamentale. Inoltre, poiché molti db non relazionali non supportano le transazioni ACID, essa non è la scelta migliore per situazioni che richiedono transazioni complesse, poiché non ne verrebbe garantita l'integrità. Inoltre, per la scalabilità e la velocità che lo caratterizzano, risulta non adatto ai contesti in cui si eseguono query complesse, comportando perdite in efficienza, soprattutto per relazioni complesse tra i dati o richieste che coinvolgono un grande carico. Infine se si è abituati ad adottare database relazionali, il passaggio da questi ai NoSQL può risultare difficile e costoso e richiedere un forte cambio di mentalità e l'acquisizione di nuove competenze[33].

3.4 Bootstrap

Bootstrap è un framework front-end open source[35]: si tratta di uno dei più popolari framework per lo sviluppo di interfacce web[36]. Esso è nato come un progetto interno a Twitter per poi diventare indipendente, e ad oggi gode di una vastissima community ed è il progetto più popolare su GitHub.

Si tratta di un tool per velocizzare e gestire al meglio l'avvio di un progetto, rendendolo responsive e semplificando la realizzazione della sua componente più estetica e visiva, basandosi sui linguaggi HTML, CSS e JavaScript.

3.4.1 Installazione

Per utilizzare Bootstrap si possono percorrere diverse strade, direttamente indicate dalla pagina ufficiale del sito[35], ovvero tramite:

- CDN
- download manuale
- package manager

Una CDN (Content Delivery Network) è una rete di server geograficamente dispersa, con lo scopo di rendere fruibili risorse web più velocemente e più vicino agli utenti finali e facilitando la distribuzione di contenuti dinamici[37]. Tramite questo metodo si può dunque usufruire di bootstrap semplicemente aggiungendo due link (uno CSS e uno JavaScript) all'interno della pagina html in cui si vuole utilizzare o, come per il progetto di questa tesi, all'interno del file base, che viene esteso tramite Django in tutti gli altri template, così da poterne usufruire nell'intero programma. In questo modo si può utilizzare il framework senza scaricare nulla localmente: i file necessari al suo funzionamento vengono caricati da un server remoto quando richiesti. Si tratta di un metodo che garantisce prestazioni e disponibilità, ma comporta anche alcuni limiti: in particolare in questo modo non è possibile far girare il progetto offline, e inoltre per eseguire personalizzazioni è necessario impiegare dei file CSS separati.

Sempre dalla home del sito ufficiale di Bootstrap [35] si può scaricare il codice sorgente: in questo modo i file, che nel caso precedente si trovavano in un server remoto, vengono interamente scaricati in locale. Inoltre il sito permette di scegliere tra due opzioni di download: scaricare la versione compilata, contenente i file

già pronti all'uso, oppure la versione sorgente, contenente dei file personalizzabili. Questa modalità, in entrambe le sue forme, è sicuramente più adatta a chi vuole avere un maggior controllo del framework.

Un package manager è un programma che permette di installare, aggiornare, rimuovere e gestire[38] pacchetti di codice. L'installazione tramite package manager, sempre indicata dalla home del sito ufficiale[35], porta i file sorgente di Bootstrap all'interno della cartella dei pacchetti del progetto. In particolare Bootstrap può essere installato tramite npm (Node Package Manager), tra i più conosciuti gestori di pacchetti JavaScript. Anche in questo caso si può procedere poi con file compilati oppure con file sorgente che rendono il tutto più personalizzabile. Come indicato dalla stessa pagina ufficiale: i package manager offrono il codice riutilizzabile (come i file JavaScript), ma non includono la documentazione o gli script di build completi.

3.4.2 Componenti

Una delle caratteristiche principali di questo framework riguarda il suo grid system (sistema a griglia): Bootstrap si basa su un sistema a griglia che suddivide la pagina html in righe e colonne, per semplificare l'organizzazione degli elementi al suo interno e permettere al programmatore di suddividere lo spazio come ritiene opportuno. In particolare suddivide la griglia su 12 colonne, identificate dalle classi da col-1 a col-12: questo significa che una volta raggiunta la grandezza massima della riga (ad esempio 2 elementi che occupano 6 colonne ciascuno), l'elemento successivo viene spostato nella riga successiva, da cui si procede nello stesso modo.

Il principale punto di forza del framework consiste nell'essere responsive: ovvero la caratteristica di un sito di adattarsi alle dimensioni dello schermo su cui viene visualizzato[39]. In un'epoca in cui accedere ad un servizio online è più frequente che accada da smartphone o comunque da schermi ridotti di quanto lo sia da computer, questa è una caratteristica fondamentale per un sito. Per rendere ciò possibile, Bootstrap sfrutta dei breakpoint che determinano le dimensioni limite dello schermo, oltre cui modificare il layout della pagina. I breakpoint di Bootstrap, con i relativi nomi della classe, sono:

- extra small (è il default): per schermi fino a 575 pixel
- small (sm): per schermi maggiori o uguali a 576 px
- medium (md): maggiori o uguali a 768 px
- large (lg): maggiori o uguali a 992 px

- extra large (xl): maggiori o uguali a 1200 px
- extra extra large (xxl): maggiori o uguali a 1400 px

Attraverso le classi "col-md-x" (con x pari ad un numero compreso tra 1 e 12) si può quindi indicare esplicitamente quante colonne (un numero pari ad x) debba occupare l'elemento in questione su schermi con più di 768 pixel. Aggiungendo allo stesso elemento una serie di classi con vari breakpoint si può determinare la sua larghezza su ogni tipo di dispositivo: Bootstrap utilizza un approccio "mobile-first", quindi prima di tutto applica le regole "col-x" (salvo sovrascrittura dei breakpoint), mentre le regole specifiche (come "col-sm-x", "col-md-x" ecc) vengono applicate solo una volta che lo schermo su cui è in uso raggiunge una larghezza pari al breakpoint (fino a quello successivo).

Le card sono un altro elemento utile: sono state impiegate nel progetto per visualizzare gli animali in maniera schematica, e similmente le associazioni presenti. Si tratta di un flessibile contenitore bootstrap che permette di implementare molte varianti. Principalmente è costituita da un "card-body", suddiviso in "card-title" e "card-text" come nel caso delle associazioni, ma può anche contenere un'immagine (come nel caso degli animali) sopra di esse attraverso la classe "card-img-top". Da questi pochi paragrafi si può dedurre quanto le classi di Bootstrap cerchino di usare un linguaggio facile ed intuitivo, permettendo di mettere le basi di un progetto rapidamente e senza dover scrivere righe di codice CSS per impostare le caratteristiche di ogni elemento, ma solo aggiungendo i nomi di classi "pre-costruite" all'interno dei tag html.

All'interno di questo progetto sono stati implementati anche i form offerti sempre da Bootstrap, che permettono di gestire facilmente gli input dell'utente con le validazioni necessarie. In particolare i form sono stati impiegati per la registrazione (sia dell'utente che dell'associazione), il login e si trovano anche all'interno delle relative pagine personali (dell'utente e dell'associazione), nascosti all'avvio della pagina, ma che diventano visibili e permettono di modificare i campi con le relative informazioni una volta premuto il pulsante di modifica. Tra le altre componenti impiegate per il sito in questione si trovano sicuramente i vari bottoni, da quello per avviare il quiz a quelli per modificare le informazioni personali, ed i messaggi di alert come il pop-up che appare quando si prova a proseguire il quiz con la domanda successiva, senza però aver selezionato una risposta per la domanda corrente.

Capitolo 4

Tecnologie per implementare il Recommendation System

Il recommendation system di questo progetto si basa sull'impiego del linguaggio di programmazione Python unito alla gestione dei dati del framework web Django. Per creare un sistema di raccomandazione, che calcolasse l'affinità su diverse metriche considerate, sono stati creati due semplici file python: uno (recommendation_system.py) per il vero e proprio calcolo delle affinità ed uno (signals.py) che attraverso l'impiego di due segnali gestisse l'avvio del calcolo al verificarsi di determinati eventi.

4.1 Architettura

Il file "recommendation_system.py" sfrutta la semplicità del linguaggio python per implementare cinque differenti metriche di similarità per il calcolo dell'affinità tra utenti e animali. Per fare ciò sono stati creati cinque rispettivi modelli che salvassero a database il punteggio per la metrica in questione per ogni utente e ogni animale. Questo processo viene avviato, come mostrato in Figura 4.1, dai segnali del file "signals.py", che avvia ognuna delle funzioni allo scatenarsi di uno dei due eventi registrati, cioè: il salvataggio di caratteristiche da parte dell'utente, che avviene ogni volta che egli completa il quiz, oppure il salvataggio delle caratteristiche da parte di un animale, che avviene quando l'animale viene registrato da parte dell'associazione o vengono modificati i valori delle sue caratteristiche.

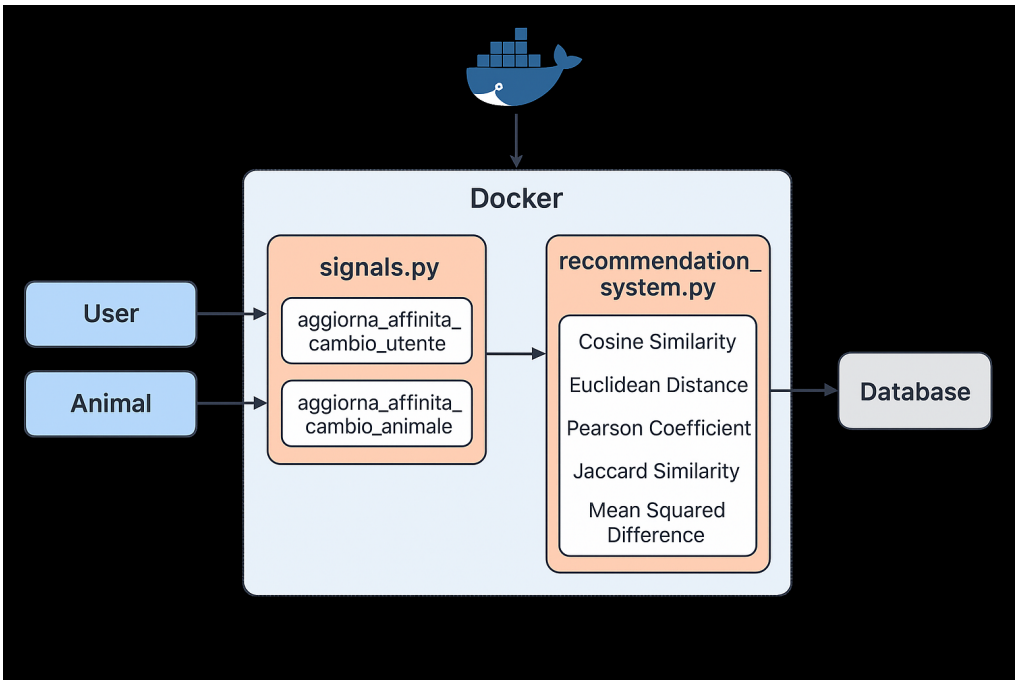


Figura 4.1: Schema del funzionamento dei signals.

Più nel dettaglio, quando ad esempio un animale viene caricato sul sito, l'associazione è tenuta a fornire anche dei valori per ciascuna sua caratteristica: il salvataggio avvia la funzione "aggiorna_affinita_cambio_animale" presente in signals.py che, a partire dall'animale in questione, avvia un ciclo su ogni utente presente a database che abbia dei valori di caratteristiche e per ognuno chiama le cinque diverse funzioni relative alle metriche di affinità, creando i relativi oggetti con i valori ottenuti.

Similmente accade al completamento del quiz da parte di un utente, che permette l'avvio della funzione "aggiorna_affinita_cambio_utente" sempre presente in signals.py e dato l'utente loggato il ciclo viene eseguito su ciascun animale che possiede almeno una caratteristica registrata e vengono ugualmente creati gli oggetti per salvare le affinità ottenute.

Il ruolo dei segnali django è dunque fondamentale: sono questi che danno il via al processo di raccomandazione tramite il segnale post_save, che viene emesso ogni volta che viene salvata o modificata a database un'istanza del modello, automatizzando così il calcolo delle metriche. Il maggior vantaggio di questo strumento risiede nella sincronizzazione automatica dei dati: infatti in questo modo si possono avere valori di affinità sempre coerenti ed aggiornati nel momento stesso in cui un utente completa un quiz o un animale viene inserito a database, senza dover

attivare nulla manualmente. Inoltre la separazione della logica dal database rende il progetto più scalabile e mantenibile.

4.2 Metriche

Per studiare le metriche dei recommendation system sono stati implementati cinque diversi algoritmi di affinità:

- Cosine Similarity
- Distanza Euclidea
- Pearson Correlation Coefficient
- Jaccard Similarity
- Mean Squared Difference

4.2.1 Cosine Similarity

La similarità del coseno misura l'angolo tra due vettori nello spazio delle caratteristiche, esso quindi favorisce la direzione in cui i vettori puntano, a discapito della loro lunghezza o dimensione [40]. Il suo valore è compreso tra -1 e 1: un valore di -1 indica che i due vettori sono esattamente opposti; lo 0 indica due vettori ortogonali, e quindi che non hanno somiglianza direzionale; un punteggio di 1 significa perfetta similarità, quindi i due vettori puntano nella stessa direzione.

La sua formula è:

$$\text{sim}_{\cos}(u, a) = \frac{u \cdot a}{\|u\| \, \|a\|}$$

Il numeratore rappresenta il prodotto scalare dei vettori u ed a, mentre il denominatore moltiplica le loro norme (le lunghezze dei vettori): al numeratore quindi viene misurato quanto essi puntino nella stessa direzione, mentre il denominatore normalizza quel valore.

Similmente in python, come mostrato in figura 4.2, dopo i controlli necessari a verificare che l'utente e l'animale considerati possiedano almeno una caratteristica, e dopo aver estrapolato le loro caratteristiche in due vettori ordinati, questo calcolo si basa su una serie di cicli for: il primo, eseguito su entrambi i vettori insieme, che

moltiplica l'i-esimo valore del vettore utente con l'i-esimo valore del vettore animale e ne somma il risultato alla variabile "sum_prod", mentre gli altri due che ciclano individualmente su un vettore sommando i quadrati dei loro valori per ottenerne la norma una volta portato il risultato finale sotto radice.

```
def calcolo_cosine_similarity(utente, animale):
    caratteristiche = Caratteristica.objects.all()
    #controllo che le caratteristiche siano presenti
    if not utente.caratteristiche_dell_utente.exists():
        return 0

    if animale.caratteristiche_dell_animale.exists():
        carat_esistenti = animale.caratteristiche_dell_animale.all()
    else:
        return 0

    vettore_utente = []
    vettore_animale = []

    for car in carat_esistenti:
        caruser = utente.caratteristiche_dell_utente.filter(caratteristica=car.caratteristica).first()
        #controllo che non sia vuoto
        valore_utente = caruser.valore if caruser else 0
        vettore_utente.append(valore_utente)
        caranimal = animale.caratteristiche_dell_animale.filter(caratteristica=car.caratteristica).first()
        valore_animale = caranimal.valore if caranimal else 0
        vettore_animale.append(valore_animale)

    #Se non ci sono caratteristiche: affinità =0
    sum_prod = 0
    #cicla su ogni coppia sommandole tra loro
    for u, a in zip(vettore_utente, vettore_animale):
        sum_prod += u * a
    #norma dei due vettori singoli
    somma_u = 0
    for u in vettore_utente:
        somma_u += u * u
    norma_utente = math.sqrt(somma_u) #aggiungere print
    print("La norma dell'utente è:", norma_utente)
    somma_a = 0
    for a in vettore_animale:
        somma_a += a * a
    norma_animale = math.sqrt(somma_a)
    print("La norma dell'animale è:", norma_animale)
    if norma_utente == 0 or norma_animale == 0:
        return 0

    similarity = sum_prod / (norma_utente*norma_animale)
    affinita = similarity*100
    return affinita
```

Figura 4.2: Codice Python della funzione per il calcolo della cosine similarity.

Questa metrica risulta ottima per contesti lessicali in cui si vuole misurare la somiglianza di contenuti, quindi ad esempio le parole simili che appaiono in due testi diversi, e non tanto la lunghezza del documento o la frequenza assoluta con cui quei termini appaiono. Similmente funziona bene anche per consigliare film, spesso categorizzati per genere o "tag", cioè insiemi di parole che ne descrivano il contenuto: in questi ambienti la cosine similarity considera solo i termini in comune (ad esempio tra due film) e ne verifica la somiglianza, che può risultare elevata anche se uno dei due film ha un vettore di tag molto più lungo dell'altro, l'importante infatti è che vi siano parole in comune e generi condivisi.

Non è però altrettanto efficiente in contesti con dati numerici, in particolar modo per vettori proporzionali ma diversi in scala, come il caso di questo progetto, poiché un vettore di caratteristiche costituito da tutti 1, secondo questa metrica, risulterebbe al 100% affine ad un vettore con tutti 10, anche se come valori si trovano agli opposti della scala, semplicemente per la loro proporzionalità.

4.2.2 Distanza Euclidea

La distanza euclidea è una misura fondamentale nel campo matematico, e calcola la distanza in linea retta tra due punti nello spazio euclideo[41]. La sua formula infatti è:

$$d_{\text{eucl}}(u, a) = \sqrt{\sum_{i=1}^{n}(u_i - a_i)^2}$$

Essa considera uno ad uno tutti i valori dei due vettori e ne calcola la distanza (prima al quadrato, per poi porre il risultato della sommatoria sotto radice).

In python, come mostrato in figura 4.3, questo è stato tradotto in un unico ciclo for che itera sulle caratteristiche esistenti, per ognuna delle quali vengono estrapolati i due valori, quello relativo all'utente e quello dell'animale, ne viene fatta la differenza ponendola poi al quadrato ed il risultato viene sommato alla variabile "somma". Alla fine del ciclo si calcola la "distanza" ponendo semplicemente la somma sotto radice quadrata. Per trasformarla in una percentuale di affinità viene prima definita la massima distanza possibile ("massima_distanza"), rappresentante due vettori, utente e animale, estremamente discordanti su tutto (posti addirittura agli estremi) ed infine si normalizza la distanza euclidea nell'intervallo [0, 100].

Si tratta di una metrica molto impiegata nell'analisi dei dati e negli algoritmi di clustering, come il K-mean, in cui i punti vengono assegnati al centroide più vicino proprio sulla base di questa misura di distanza[41]. Questa metrica inoltre soddisfa la disuguaglianza triangolare, garantendo misure coerenti e affidabili in spazi multidimensionali.

Uno dei limiti della distanza euclidea riguarda la scala di valori utilizzata: in particolare questa misura è sensibile al range di valori utilizzati, per questo i dati devono prima essere normalizzati. Per quanto riguarda il progetto in questione, tutte le caratteristiche hanno la stessa scala e possono assumere valori compresi in un range 0-10, quindi non è stato necessario implementare ulteriori normalizzazioni.

```python
def distanza_euclidea(utente, animale):
    caratteristiche = Caratteristica.objects.all()
    #11 caratteristiche che arrivano massimo a 10 pt
    if not utente.caratteristiche_dell_utente.exists():
        return 0
    if animale.caratteristiche_dell_animale.exists():
        carat_esistenti = animale.caratteristiche_dell_animale.all()
    else:
        return 0
    contatore = carat_esistenti.count()
    massima_distanza = 10 * math.sqrt(contatore)
    somma = 0
    for c in carat_esistenti:
        cutente = utente.caratteristiche_dell_utente.filter(caratteristica=c.caratteristica).first()
        canimale = animale.caratteristiche_dell_animale.filter(caratteristica=c.caratteristica).first()

        #controllo post
        if not cutente or not canimale:
            continue

        differenza = cutente.valore - canimale.valore
        quadrato = differenza ** 2
        somma += quadrato
    distanza = math.sqrt(somma)
    affinita = (1 - distanza / massima_distanza) * 100
    return affinita
```

Figura 4.3: Codice Python della funzione per il calcolo della distanza euclidea.

4.2.3 Pearson Correlation Coefficient

Il coefficiente di Correlazione di Pearson è una delle tecniche statistiche più diffuse per misurare la correlazione tra due variabili, ovvero la tendenza di due variabili di variare insieme[42]. I valori che può assumere vanno da -1, che indica perfetta correlazione negativa, ovvero quando una variabile aumenta l'altra diminuisce, fino a 1, cioè perfetta correlazione positiva (aumentano o diminuiscono insieme) passando per 0, cioè l'assenza di correlazione[43]. La sua formula è:

$$r = \frac{\sum_{i=1}^{n}(u_i - \bar{u})(a_i - \bar{a})}{\sqrt{\sum_{i=1}^{n}(u_i - \bar{u})^2}\,\sqrt{\sum_{i=1}^{n}(a_i - \bar{a})^2}}$$

Il numeratore misura la variazione che i due vettori hanno insieme, mentre il denominatore normalizza quel valore per le loro deviazioni standard in modo che il risultato sia compreso tra -1 e 1.

In python, figura 4.4, si sono estratti i due vettori di caratteristiche: quello dell'utente e quello dell'animale. Una volta ottenuti questi, la funzione calcola le due medie dei vettori e con un ciclo for sui due vettori si ottiene il numeratore, come somma incrementata ciclo per ciclo del prodotto tra il valore utente (a cui viene sottratta la sua media) e il rispettivo valore animale (ottenuto allo stesso modo); così si ottiene la covarianza non normalizzata dei due vettori. Successivamente con altri due cicli for si ottiene il denominatore, costituito dalle deviazioni standard. A

questo punto viene calcolata la variabile "pearson" che è esattamente la formula sopra citata, con valori compresi tra -1 e 1. La variabile "affinita" che poi viene restituita dalla funzione serve a spostare il range di valori da [-1, 1] a [0, 100] tramite: una traslazione che porta il range a [0, 2] aggiungendo 1 alla variabile "pearson", un ridimensionamento dividendo per 2 così da ottenere un range [0, 1], ed infine la conversione in percentuale moltiplicando per 100.

```
71
72  def calcolo_pearson(utente, animale):
73
74      if not utente.caratteristiche_dell_utente.exists():
75          return 0
76
77      if animale.caratteristiche_dell_animale.exists():
78          carat_esistenti = animale.caratteristiche_dell_animale.all()
79      else:
80          return 0
81
82      vettore_utente = []
83      vettore_animale = []
84
85      for car in carat_esistenti:
86          valore_a = car.valore
87          car_u = utente.caratteristiche_dell_utente.filter(caratteristica=car.caratteristica).first()
88          valore_u = car_u.valore if car_u else 0
89          vettore_animale.append(valore_a)
90          vettore_utente.append(valore_u)
91
92      media_a = sum(vettore_animale) / len(vettore_animale)
93      media_u = sum(vettore_utente) / len(vettore_utente)
94      numeratore = sum((u - media_u) * (a - media_a) for u, a in zip(vettore_utente, vettore_animale))
95      denom_u = 0
96      denom_a = 0
97      for u in vettore_utente:
98          denom_u += (u - media_u) ** 2
99      for a in vettore_animale:
100         denom_a += (a - media_a) ** 2
101     denominatore_u = math.sqrt(denom_u)
102     denominatore_a = math.sqrt(denom_a)
103
104     if denominatore_u == 0 or denominatore_a == 0:
105         return 0
106
107     pearson = numeratore / (denominatore_a * denominatore_u)
108     affinita = (pearson + 1)/ 2 * 100
109     return affinita
```

Figura 4.4: Codice Python della funzione per il calcolo del coefficiente di correlazione di Pearson.

Data la sua formula questo coefficiente normalizza i dati già da solo, dividendo la covarianza per il prodotto delle deviazioni standard, rendendo quindi questa metrica ottima per confrontare valori provenienti da scale differenti. Inoltre è molto facile da interpretare ed è ottimo per far emergere relazioni lineari. Esso però dipende fortemente da variabili centrate sulla media, quindi non è troppo indicato per distribuzioni asimmetriche o dati mancanti.

Nel contesto di questo progetto, in cui si vuole calcolare la compatibilità tra utente e animale, questo coefficiente può avere dei limiti in quanto: può essere

sensibile a dei valori molto alti o molto bassi dei vettori che possono alterare il coefficiente e catturando il pattern e l'andamento dei due vettori, non si concentra abbastanza sulla reale distanza che essi possono avere.

4.2.4 Jaccard Similarity

La similarità, o l'indice, di Jaccard indica gli elementi in comune tra due insiemi rispetto al totale degli elementi presenti. Infatti è calcolata come l'intersezione dei due insieme divisa per la loro unione. La sua formula è:

$$\text{sim}_{\text{Jaccard}}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Al numeratore quindi si trovano tutti gli elementi appartenenti esclusivamente ad entrambi i gruppi, mentre al denominatore vengono considerati tutti gli elementi singoli presenti in entrambi gli insiemi. Questa metrica tipicamente utilizza valori binari (0, 1) per indicare la presenza o l'assenza di un elemento, ed il risultato è un numero compreso tra 0 ed 1 dove il valore che più si avvicina ad 1 indica la maggiore somiglianza tra i vettori[44].

I vantaggi di questo indice includono sicuramente la semplicità nell'implementarlo; inoltre esso non è influenzato dalle dimensioni del set di dati e si applica bene a molti contesti. Valuta però la presenza e l'assenza ed è quindi più adatto a quel tipo di contesti.

Nel progetto in questione vi sono una serie di caratteristiche, uguali per utenti ed animali, con valori appartenenti ad un range [0, 10]. Già da questo emerge il primo problema, trasformato in una perdita di informazioni. Infatti, per poter eseguire il calcolo di questa metrica sul progetto in esame come mostrato in figura 4.5, è stata eseguita una forte semplificazione: all'interno di un ciclo for che itera su tutte le caratteristiche, per ogni valore di utente e animale maggiore di una soglia scelta pari a 5 è stato assegnato un valore pari ad 1, mentre per tutti gli altri valori (quindi minori o uguali a 5) il nuovo valore assegnato è pari a zero. Infine, a costituire il numeratore, non vi sono solo i valori pari ad uno, che tipicamente indicano la presenza di un elemento (o caratteristica) ma, in questo caso, il numeratore viene incrementato di uno ogni volta che il valore dell'utente è pari al valore dell'animale, il ché indica che quei due valori appartengono allo stesso sottoinsieme creato appositamente per questa funzione, cioè entrambi tra [6, 10], oppure entrambi tra [0, 5]. È evidente come questa semplificazione comporti una notevole perdita di informazioni: due elementi possono essere ritenuti molto affini semplicemente se hanno valori alti o bassi delle stesse caratteristiche, senza considerare se all'interno di quel range si trovino comunque distanti o addirittura agli estremi e senza considerare

```python

def calcolo_jaccard(utente, animale):
    if not utente.caratteristiche_dell_utente.exists():
        return 0
    if animale.caratteristiche_dell_animale.exists():
        caratteristiche = animale.caratteristiche_dell_animale.all()
    else:
        return 0

    #intersezione/unione
    #modificata con denominatore = tutte le caratteristiche dell'animale presenti
    #numeratore quelle in cui sono d'accordo

    numeratore = 0
    denominatore = len(caratteristiche)

    for car in caratteristiche:
        val_u = utente.caratteristiche_dell_utente.filter(caratteristica=car.caratteristica).first()
        val_a = animale.caratteristiche_dell_animale.filter(caratteristica=car.caratteristica).first()
        if val_u.valore > 5:
            u = 1
        else:
            u = 0
        if val_a.valore > 5:
            a = 1
        else:
            a = 0
        if u == a:
            numeratore += 1

    risultato = numeratore / denominatore
    affinita = risultato * 100
    return affinita
```

Figura 4.5: Codice Python della funzione per il calcolo della Jaccard similarity.

nemmeno i casi limite in cui vi sono valori vicini ma che salgono e scendono di poco rispetto alla soglia.

4.2.5 Mean Squared Difference

La differenza quadratica media (MSD), come indica il nome, calcola la differenza quadratica media tra le coppie di elementi[45]. La sua formula è:

$$\text{msd}(u, a) = \frac{1}{n} \sum_{i=1}^{n} (u_i - a_i)^2$$

Questa metrica si concentra sulla differenza di valori tra i due vettori utente e animale, enfatizzando molto quando essi differiscono, grazie all'elevamento al quadrato.

Rispetto alle metriche analizzate fino a qui, essa può essere paragonata alla distanza euclidea, in quanto si concentra sui reali valori dei vettori, e non, come nel caso della Cosine Similarity o del coefficiente di Pearson, alla direzione o alla

correlazione. È una metrica molto intuitiva e semplice da implementare, e si presta bene al caso in questione in cui i dati si trovano su una stessa scala uniforme.

Il codice python per implementare questa funzione, figura 4.6, dimostra la sua semplicità: il numeratore viene calcolato attraverso un ciclo for che itera su tutte le caratteristiche e per ognuna di esse eleva al quadrato la differenza tra il valore dell'utente ed il corrispettivo valore dell'animale, per poi sommare ognuno di questi risultati nella variabile "numeratore", mentre il denominatore è semplicemente pari al numero di caratteristiche. Proprio come per il caso della distanza euclidea, anche questa metrica indica un valore di distanza, e quindi cresce all'aumentare della "non-affinità" tra i due elementi analizzati: quindi come per quel caso anche qui è stato manipolato il risultato per ottenerne una percentuale di "affinità".

```
144  def calcolo_msd(utente, animale):
145      #Mean Squared Difference
146      if not utente.caratteristiche_dell_utente.exists():
147          return 0
148      if animale.caratteristiche_dell_animale.exists():
149          caratteristiche = animale.caratteristiche_dell_animale.all()
150      else:
151          return 0
152
153      denominatore = len(caratteristiche)
154      numeratore = 0
155
156      for c in caratteristiche:
157          val_u = utente.caratteristiche_dell_utente.filter(caratteristica=c.caratteristica).first()
158          val_a = animale.caratteristiche_dell_animale.filter(caratteristica=c.caratteristica).first()
159          differenza = (val_u.valore - val_a.valore) ** 2
160          numeratore += differenza
161
162      msd = numeratore / denominatore
163
164      affinita = (1 - (msd/100)) * 100
165      return affinita
```

Figura 4.6: Codice Python della funzione per il calcolo della Mean Squared Difference.

Capitolo 5

La piattaforma a livello funzionale

Per il progetto di questa tesi sono stati impiegati gli strumenti spiegati a livello più teorico nei capitoli precedenti, al fine di ottenere un sito web responsive, che potesse consigliare abbinamenti uomo-animale il più possibile duraturi nel tempo. Per ottenere questo risultato sono stati studiati i tipi di sistemi di raccomandazione e le metriche possibili, ed è stato creato un quiz da sottoporre agli utenti della piattaforma con domande pensate per mettere in luce le caratteristiche che possono rappresentare degli "ostacoli" nel rapporto tra un essere umano ed il suo fedele amico a quattro zampe.

5.1 Struttura dei dati

I modelli di questo progetto si trovano tutti nel file nuovimodelli/models.py. Essi vanno a popolare una serie di tabelle a database che possono essere graficamente rappresentate dal diagramma ER (Entità-Relazioni) di figura 5.1.

Il diagramma ER è uno strumento grafico utilizzato per rappresentare come gli elementi di un database si relazionano tra di loro[46]. I suoi componenti principali sono:

- Entità: rappresentate da rettangoli, corrispondono agli oggetti principali del database; possono essere persone fisiche ma anche concetti, oggetti o avvenimenti che necessitano di registrare a database delle informazioni a loro

riguardanti.

- Attributi: sono caratteristiche che definiscono l'entità; tra queste vi sono le chiavi, necessarie a identificare univocamente un'istanza di quel modello, ma anche proprietà dell'oggetto in questione, che possono tradursi in attributi semplici o derivati (calcolati a partire da altri valori).
- Relazioni: rappresentate dalle linee che collegano tra di loro le varie entità, poiché indicano proprio come queste sono associate tra loro; esse hanno una cardinalità, che mostra quante istanze di un'entità possono riferirsi a quante istanze dell'altra entità.

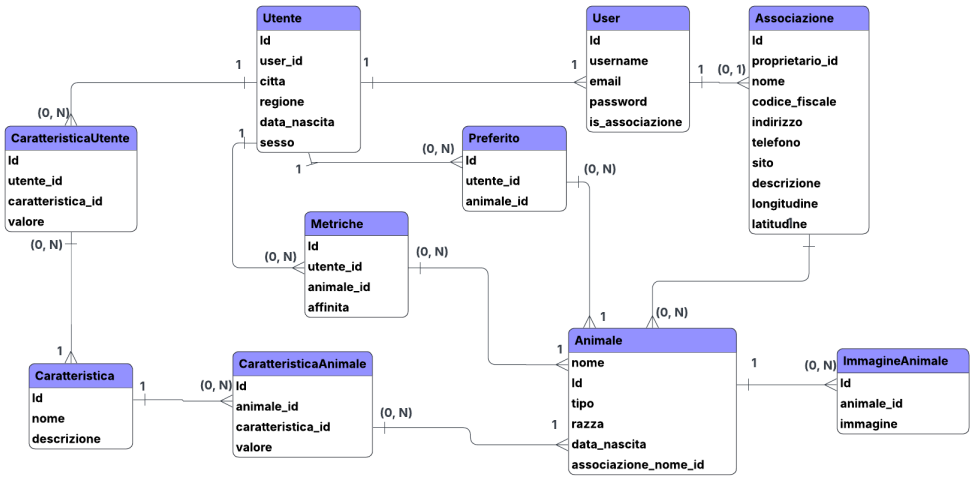


Figura 5.1: Diagramma ER che mostra la struttura del database del progetto.

Come rappresenta il diagramma in figura 5.1, in questo progetto sono state create innanzitutto due classi per gli utenti: ci si è appoggiati alla classe User di Django, legata con una relazione 1 ad 1 alla classe Utente. Questo indica che un utente che si registra sulla piattaforma avrà automaticamente un proprio profilo Utente, associato univocamente ad un proprio profilo User. Questa scelta è stata condizionata dalla vastità di opportunità offerte dal framework Django, così da semplificare l'implementazione del login e dell'autenticazione, che avviene sulla base del profilo User. Solo una volta verificata la validità delle credenziali, il sistema ricerca il profilo Utente associato a quel profilo User e l'utente entra nella piattaforma potendo visualizzare tutti i suoi dati.

Similmente avviene per le Associazioni, anche queste associate ad un user, ma in quel caso si tratta del proprietario: all'Associazione infatti può accedere solo il proprietario, tramite il proprio profilo user, o meglio, tramite il codice fiscale

dell'associazione e la password del suo profilo privato User. La relazione che lega quindi queste due entità è anche in questo caso univoca, ma viene rappresentata da 1 e (0, 1) poiché: ogni Associazione ha un unico proprietario (associazione con User ad 1), ma non è detto che ogni utente abbia un'associazione, tra gli user vi sono utenti normali che non fanno parte delle associazioni, e utenti che sono proprietari di un'associazione, per questo la cardinalità in questo caso è (0, 1), poiché quella relazione può non esistere per alcuni, ma dove c'è è univoca.

Un'altra entità fondamentale della piattaforma è data dal modello Animale: essi possono essere registrati dalle Associazioni; ogni animale registrato può appartenere ad una sola Associazione (cardinalità pari ad 1) mentre un'Associazione può avere molti Animali, oppure nessuno se non ne ha ancora registrato nessuno (cardinalità quindi che va da 0 ad N).

Il fulcro poi di tutto il progetto, in realtà, è costituito dalle metriche di affinità. Nel progetto sono state implementate cinque differenti metriche, ognuna delle quali possiede una sua propria classe e tabella a database, come riportato nella figura 5.2.

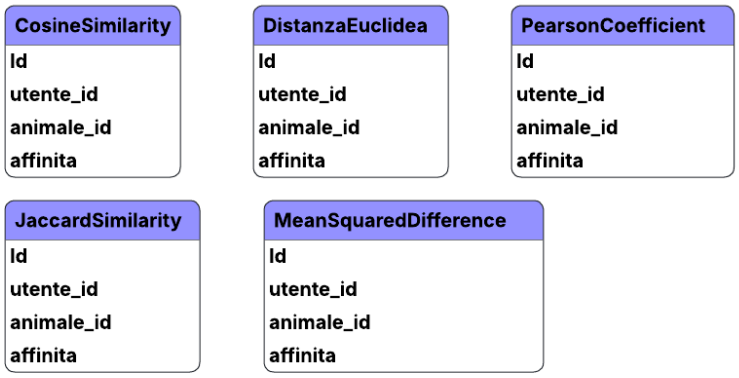


Figura 5.2: Classi relative alle metriche di affinità - estensione della classe "Metriche"

Per una visualizzazione più semplice ed immediata, in figura 5.1 è stata impiegata un'unica classe fittizia, chiamata "Metriche", che rappresentasse il funzionamento di ognuna delle cinque reali tabelle rappresentate nel diagramma successivo. Le reali tabelle a database sono: CosineSimilarity, DistanzaEuclidea, PearsonCoefficient, JaccardSimilarity e MeanSquaredDifference. Ognuna di loro può essere rappresentata dalla tabella fittizia "Metriche", ed infatti:

- ognuna di loro possiede come attributi un id di ogni istanza, le chiavi esterne

con le classi utente e animale e il relativo valore derivato dal calcolo dell'affinità;

- ognuna di loro ha una relazione con le classi utente e animale secondo cui ogni oggetto di quella metrica è legato ad uno e un solo utente e similmente ad un unico animale, mentre entrambi i rami che raggiungono l'oggetto in questione hanno cardinalità (0, N) in quanto ogni utente può non avere nessuna metrica attuale se non ha ancora svolto il quiz fino ad avere N valori (dove N è pari al numero di animali registrati e con le caratteristiche), allo stesso modo funziona per gli animali.

Quando viene dunque calcolata l'affinità, vengono creati cinque diversi oggetti per ogni relazione utente-animale. Questo approccio che è stato implementato ovviamente era solo una delle scelte possibili, similmente si poteva procedere implementando solo una classe di "metriche", come rappresentato in figura 5.1, ma con una variabile in più riguardante il tipo di metrica che era stata implementata. Si è scelto di procedere in questo modo per ottenere una maggior immediatezza e chiarezza nel visualizzare i risultati, tenendo tutto separato, e anche in funzione dell'obiettivo finale di raggiungere, grazie ad un'analisi, un'unica metrica che sia la più affidabile possibile tra quelle proposte.

5.2 Registrazione

Come mostrato dalla struttura del database, all'interno di questa piattaforma sono previsti due tipi di "utilizzatori" attivi: gli utenti e le associazioni; ma sono principalmente tre gli oggetti che possono essere registrati, racchiudendo quindi anche gli animali. Ognuno di loro viene salvato a database in maniera differente. In questa sezione viene mostrato il caso d'uso relativo alla loro registrazione. La più simile registrazione avviene tra utenti e associazioni, in cui cambiano solo alcuni campi del form ed il numero di oggetti creato.

5.2.1 Registrazione Utente semplice

Quando si accede al form di registrazione della piattaforma viene presentato, come primo campo del form, un menù a discesa (dropdown menu) con due opzioni: "Utente" e "Associazione". Selezionando l'opzione "utente" (quella di default già selezionata all'avvio della pagina) si può visualizzare un form di dimensioni abbastanza ridotte, mostrato in Figura 5.3. L'utente che desidera registrarsi deve quindi compilare i suoi campi immettendo: il proprio nome e cognome (in due appositi

campi di testo distinti), la mail e la password che servono come credenziali di accesso alla piattaforma, la data di nascita (selezionabile dal calendario che appare), il sesso (un altro menù a discesa con le opzioni "Maschio", "Femmina" e "Non specificato"), ed infine la propria città e regione in altre due caselle di testo. Una volta terminata la compilazione del form, l'utente può premere il bottone di "Conferma" e vengono creati due oggetti, uno Utente ed uno User, a lui associati. A questo punto l'utente vedrà il messaggio "Registrazione completata con successo" e la piattaforma lo rimanderà automaticamente alla pagina di login, dalla quale potrà accedere usando la propria mail e password.

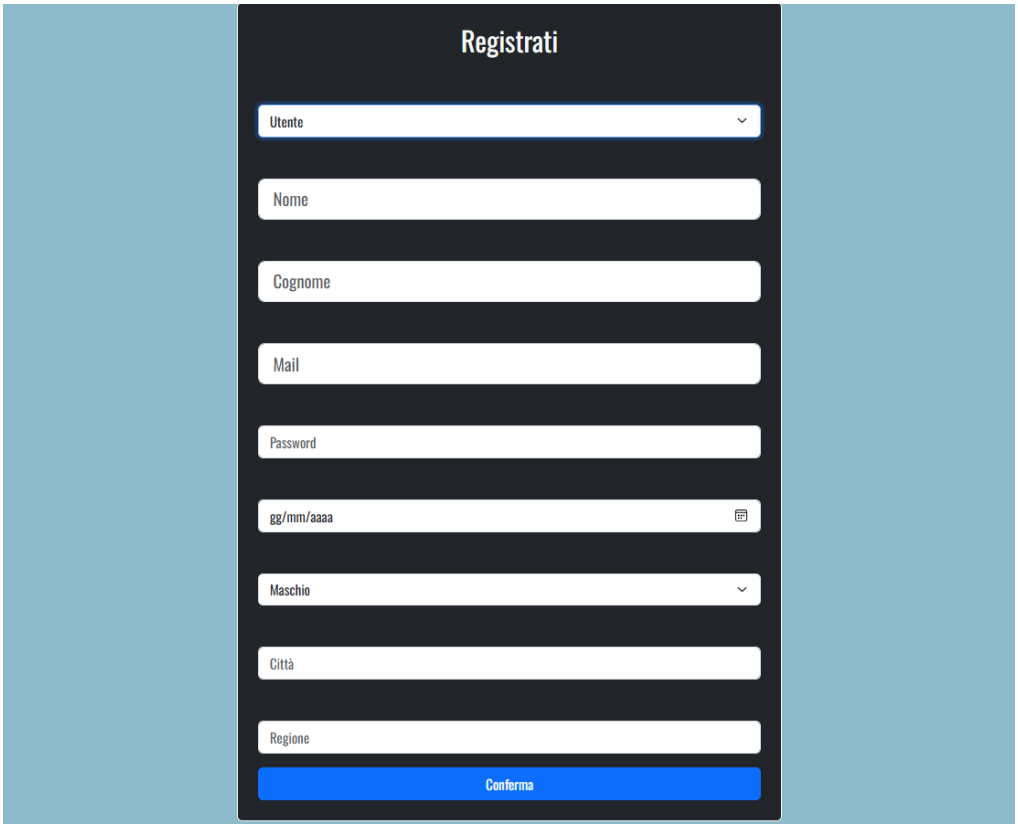


Figura 5.3: Schermata di registrazione per l'utente.

Se però, durante la registrazione, l'utente dimenticasse di compilare uno tra i campi "nome", "cognome", "email" o "password" verrebbe visualizzato un messaggio di errore contenente "Compila tutti i campi correttamente!" e l'utente si ritroverebbe davanti alla pagina di registrazione pulita. Similmente vi è anche un controllo più specifico sulla mail, che confronta la mail inserita nel form con tutte le mail presenti a database per verificare che non sia già stata utilizzata; se così fosse, il messaggio a cui si troverebbe davanti l'utente sarebbe "Email già registrata" e dovrebbe ricominciare con la compilazione del form, che gli verrebbe riproposto.

5.2.2 Registrazione Associazione e Proprietario

La registrazione dell'Associazione, pur presentando campi simili a quella dell'utente, è in realtà concettualmente diversa. L'Associazione viene registrata a database dal suo proprietario, che la registrerà insieme alla registrazione del proprio account utente personale, a tal punto legate tra di loro che questi due oggetti avranno la stessa password in comune.

Il proprietario che vuole procedere alla registrazione accede alla stessa pagina a cui arrivava anche l'utente semplice precedentemente descritto. Dal form in questione, il proprietario selezionarà però "Associazione" nel primo campo, permettendo così alla pagina di mostrare i campi inizialmente nascosti, relativi appunto a questo oggetto, Figura 5.4. A questo punto l'utente in questione procederà con l'immissione dei dati richiesti dal form, prima relativi a se stesso e poi alla sua associazione. I campi relativi a lui sono gli stessi presentati per la "Registrazione Utente Semplice", e quindi il proprietario dovrà inserire: nome, cognome, mail, password, data di nascita, sesso, città e regione. Per quanto riguarda l'associazione vengono richiesti in ordine:

- nome dell'associazione;
- codice fiscale dell'associazione;
- indirizzo;
- telefono;
- sito web (opzionale).

Una volta completato il form, il proprietario può procedere premendo il bottone di "Conferma".

A questo punto valgono tutti i controlli dell'utente semplice, a cui ne vengono aggiunti altri relativi all'Associazione. Infatti, se l'utente ha dimenticato di compilare uno tra i campi "nome", "cognome", "email" e "password" riferiti a se stesso o uno tra i campi "nome_associazione", "codice_fiscale", "indirizzo" e "telefono" per l'associazione, vedrà un messaggio che segnala il problema, in particolare "Compila tutti i campi correttamente" nel primo caso e "Compila tutti i campi" nel secondo, e sarà riportato sulla pagina di registrazione con il form pulito. Per quanto riguarda la possibile duplicazione di un oggetto già registrato vale sempre il controllo sulla mail implementato anche per l'utente semplice, che verifica la mail inserita nel form con quelle presenti a database ed eventualmente mostra il messaggio "Email già registrata" permettendo di ricompilare il form da capo.

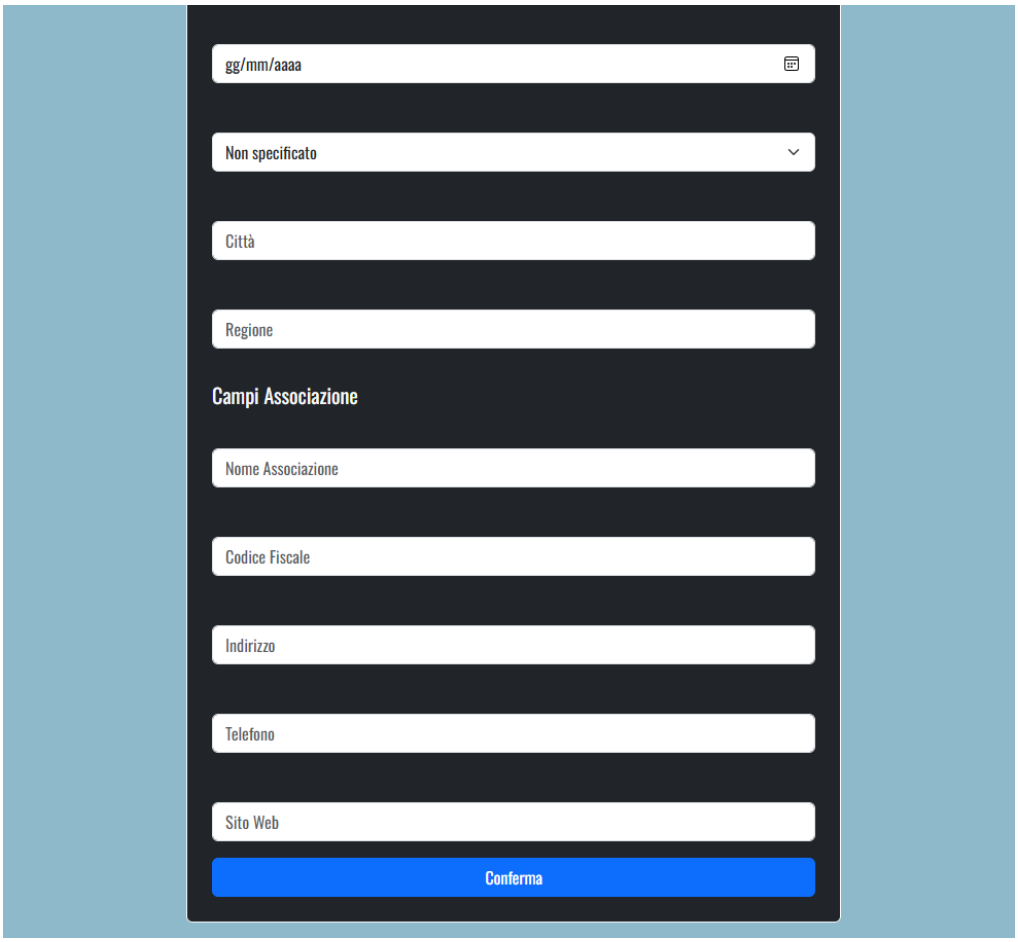


Figura 5.4: Campi aggiuntivi per la registrazione dell'Associazione.

Se invece vengono immesse tutte le informazioni necessarie e valide, in questo caso gli oggetti che vengono creati a database sono tre: un user ed un utente, relativi al proprietario dell'associazione, legati alla sua email e password, ed un oggetto associazione, rappresentante appunto l'associazione, e legato al codice fiscale di questa (al posto della mail per gli utenti) ma anche alla stessa password del proprietario. A questo punto la pagina mostra il messaggio "Registrazione associazione effettuata con successo" prima di spostarsi sulla pagina di login, dalla quale il proprietario potrà scegliere se accedere al suo profilo (e insieme a quello dell'associazione) selezionando il tipo "Utente" nel menù a discesa e procedendo con l'inserimento della sua mail e password, oppure selezionando "Associazione" e inserendo il codice fiscale dell'associazione e sempre la sua password.

5.2.3 Registrazione Animale

La registrazione di un animale può avvenire solo da parte dell'Associazione che lo ha in carico e quindi dal profilo di quest'ultima. L'Associazione, dopo aver eseguito l'accesso alla piattaforma, può spostarsi dal menù laterale a sinistra sulla pagina "I nostri animali", relativa appunto agli animali di quell'associazione. Da questa pagina può visualizzare tutti gli animali che ha già registrato a database, e selezionando il bottone "Aggiungi un animale", vedrà apparire una modale in cui inserire tutte le informazioni necessarie alla registrazione dell'animale in questione.

All'interno del form di questa modale si chiederà all'utente di inserire i dati relativi ai seguenti campi:

- Nome dell'animale.
- Tipo di animale, grazie ad un menù a discesa con le tre seguenti opzioni: "Cane", "Gatto" o "Altro".
- Razza.
- Immagini: vi è la possibilità di caricare delle immagini relative all'animale in questione, di cui la prima verrà usata come copertina della sua card; si tratta di un campo opzionale, e se non vengono caricate immagini ne viene usata una di default come copertina.
- Infine vi è l'elenco di caratteristiche, per ognuna delle quali l'associazione è chiamata a fornire un punteggio da 0 a 10 al fine di descrivere quell'animale il meglio possibile.

Infine l'utente che interagisce in nome dell'associazione, dopo aver completato il form, può premere il bottone "Salva" e potrà visualizzare nuovamente la pagina relativa a "I nostri animali", vedendo in tempo reale una nuova card, riferita all'animale appena registrato.

5.3 I template del sito

Una volta effettuato il login, come mostrato in Figura 5.5, secondo le credenziali esplicitate nei paragrafi relativi alla registrazione, l'utente o l'associazione si trovano davanti alla propria pagina personale, con un menù verticale sul lato sinistro la cui maggior parte delle voci sono in comune ad entrambi.

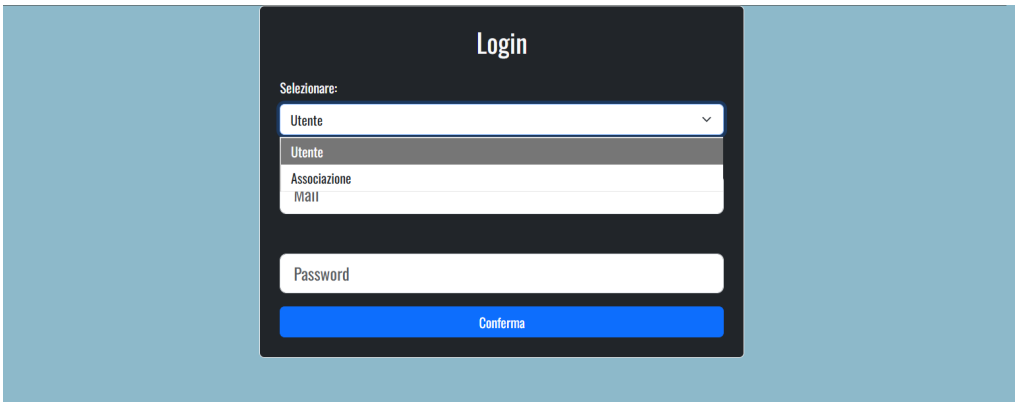


Figura 5.5: Schermata del Login.

La navbar verticale permette di navigare tra le pagine del sito, le cui etichette sono mostrate sotto il logo della piattaforma. Essa è nascosta per la pagina di registrazione e di login, e presente per tutte le altre. Per gli utenti non ancora registrati permette di selezionare solo alcune pagine, in particolare: Home, Animali, Associazioni, Login e Registrati. Invece, per utenti autenticati le voci "Login" e "Registrati" lasciano il posto alla voce "Logout" che permette la disconnessione, e vengono mostrate anche le pagine "Quiz" e "Profilo Personale". Per le Associazioni a queste si aggiungono le pagine "I nostri animali" e "Pagina Associazione".

5.3.1 Pagine Personali

La pagina personale dell'utente, visibile in Figura 5.6, presenta tutte le informazioni dell'utente appena loggato, con in fondo un bottone "Modifica", che una volta premuto rende le label modificabili e rende visibile il bottone "Salva" per salvare appunto eventuali modifiche. Questa è la pagina di atterraggio al login di un utente, ma è anche una pagina visibile dall'associazione (anche se non quella di atterraggio) e selezionabile dal menù principale sul lato sinistro dello schermo. Nel caso dell'Associazione, Figura 5.7, essa riporta tutte le informazioni relative al proprietario ed in generale è raggiungibile dalla navbar tramite l'etichetta "Profilo Personale". In fondo alla pagina l'utente può vedere anche le card relative ai suoi animali preferiti, ovvero quelli sui quali ha cliccato il pulsante a forma di cuore.

Quando invece il login avviene per conto di un'Associazione, esso rimanda alla pagina personale dell'associazione, visibile solo a questo tipo di oggetti o ad un oggetto utente legato ad un'associazione in quanto il proprietario. La voce della navbar relativa a "Pagina Associazione" infatti non viene neanche visualizzata da utenti semplici. Similmente alla pagine dell'utente, questo template permette

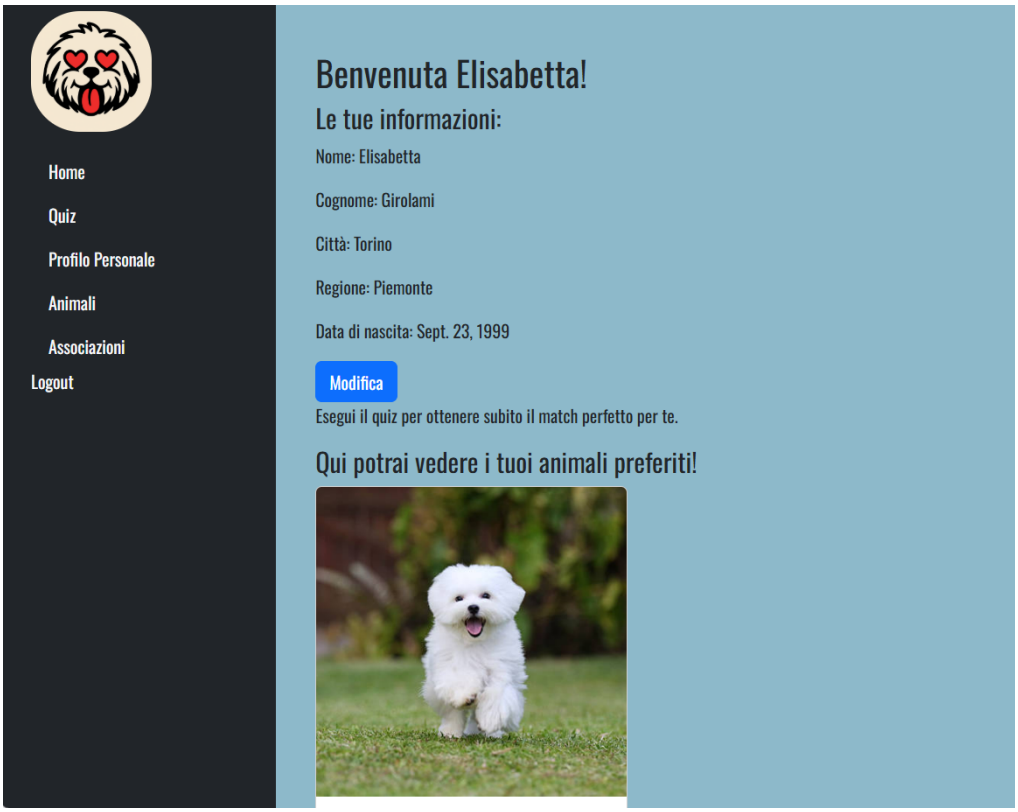


Figura 5.6: Schermata della pagina personale dell'utente.

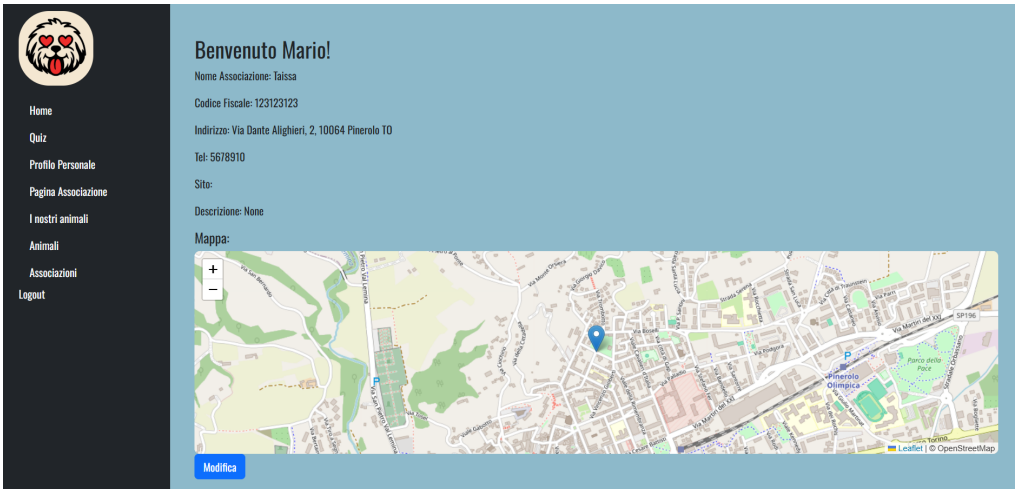


Figura 5.7: Schermata della pagina personale dell'associaizone.

di visualizzare tutte le informazioni relative all'associazione, e tramite l'apposito

pulsante permette anche di modificarle e poi salvarle. Dopo tutte le informazioni viene visualizzata una mappa, dove possibile, relativa all'indirizzo immesso per l'Associazione.

Anche tutti gli animali registrati godono di una loro pagina personale, o sarebbe meglio definirla "scheda privata", visibile in Figura 5.8. In questa pagina si può visualizzare, per gli animali che ne hanno almeno una, l'immagine dell'animale in questione (altrimenti nulla); successivamente vi sono informazioni relative alla sua razza e l'elenco delle sue caratteristiche, i cui valori per ognuna di esse sono rappresentati da delle stelle piene colorate su un massimo di dieci stelle (le restanti stelle per arrivare al massimo di dieci sono lasciate vuote).

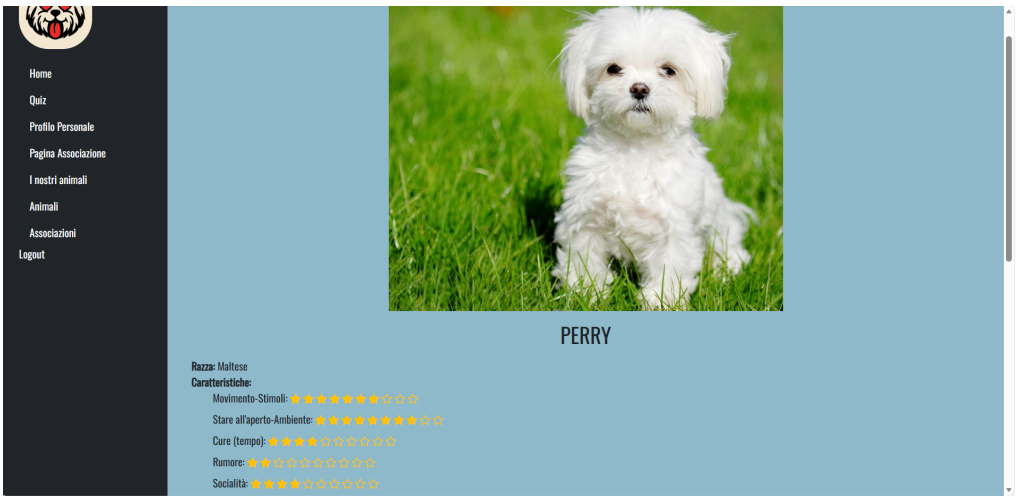


Figura 5.8: Prima parte della scheda personale dell'animale.

Alla fine della sua pagina viene riportato anche l'indirizzo dell'Associazione in cui si trova e la mappa, dove possibile, dell'associazione, come mostrato in Figura 5.9; dove non vi sono abbastanza informazioni da poter visualizzare la mappa (perché l'indirizzo magari non è completo), viene visualizzata la scritta "mappa non disponibile". Questa scheda relativa all'animale però, diversamente dalle due pagine precedenti, non è raggiungibile dalla navbar laterale, ma solo tramite il link presente sulla card dell'animale in questione, che si può vedere dalla pagina relativa a tutti gli animali presenti a database e corrispondente alla voce "Animali" del menù.

5.3.2 Animali e Associazioni

Due pagine visibili a tutti gli utenti, anche non registrati sulla piattaforma, sono "Animali" e "Associazioni". Esse riportano una serie di card, una per ogni animale

Figura 5.9: Seconda parte della scheda personale dell'animale.

nel primo caso, e per ogni associazione nel secondo.

Nella pagina rappresentante tutti gli animali, mostrata in Figura 5.10, vi sono due filtri in alto che permettono di selezionare e visualizzare le card relative al tipo, per il primo menù a discesa, o alla razza per il secondo filtro. Inoltre vi è un menù riportante anche le cinque metriche implementate, e che permette di ordinare la visualizzazione degli animali in ordine decrescente di affinità secondo la metrica selezionata. Successivamente nella pagina vi sono tutti gli animali disponibili, e per ognuno di essi la relativa card mostra: l'immagine (dove è stata caricata, altrimenti l'immagine di default), un pulsante a forma di cuore cliccabile, che se selezionato si riempie (diventando rosso) e aggiunge quell'animale alla lista dei preferiti dell'utente loggato, un link che porta alla scheda dell'animale, il tipo e la razza dell'animale e successivamente l'elenco delle metriche di affinità con i valori risultanti per ognuna di esse. Nel caso in cui all'animale non fossero ancora stati assegnati dei valori per le caratteristiche (o l'utente non avesse completato il quiz) e quindi non fosse possibile calcolare l'affinità, viene riportato l'elenco di metriche con la scritta "non calcolata" (ad esempio "Distanza Euclidea non calcolata).

Similmente nella pagina "Associazioni", come mostrato in Figura 5.11, si possono visualizzare l'elenco di associazioni registrate a database, ognuna con la propria card. Le card in questo caso mostrano: il nome dell'associazione, la sua descrizione (inseribile dalla pagina personale dell'associazione), l'indirizzo, il recapito telefonico, il sito web e, dove l'indirizzo è abbastanza completo da permetterne la geolocalizzazione, si può visualizzare anche la mappa, altrimenti viene riportato "mappa non disponibile".

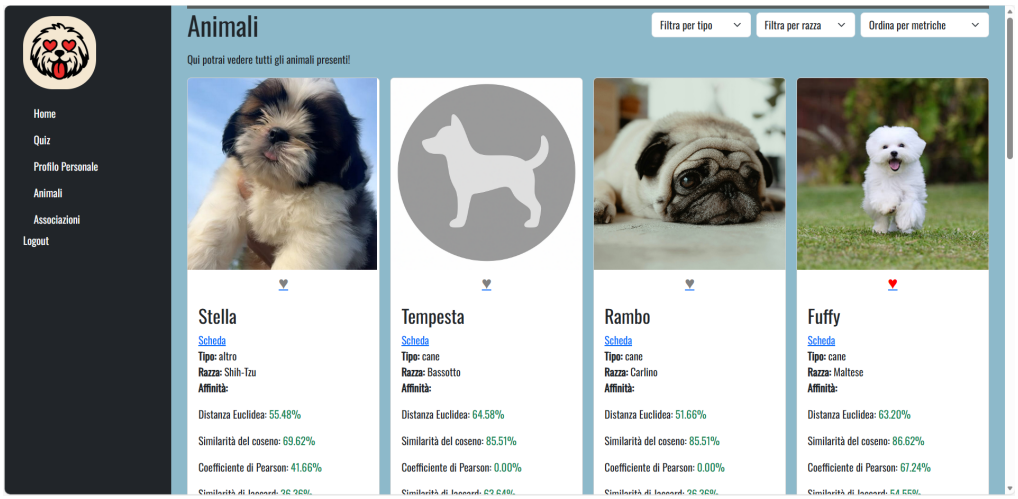


Figura 5.10: Schermata relativa alla pagina "animali".

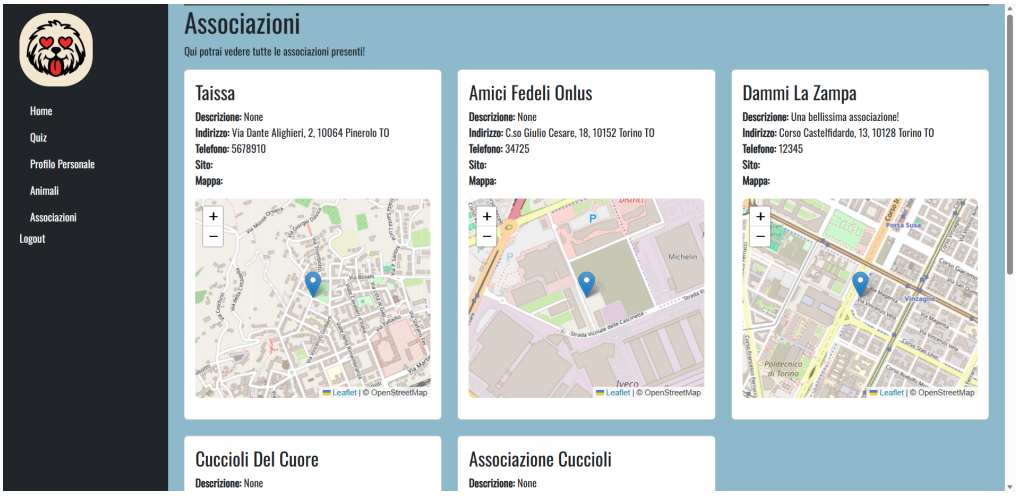


Figura 5.11: Schermata relativa alla pagina "associazioni".

5.3.3 Quiz

La pagina "Quiz", visibile solo agli utenti registrati sulla piattaforma, all'apertura presenta un solo bottone "Quiz", per avviare le domande, come mostrato in Figura 5.12. Una volta premuto, la pagina mostra un riquadro centrale alla pagina, con la prima domanda, e una progress bar che si aggiorna man mano che si procede, e che indica il numero della domanda che si è raggiunto, partendo da 1/13, Figura 5.13.

Sul fondo del riquadro vi sono due bottoni, "Indietro" (nascosto per la prima domanda) e "Prossima domanda". Le domande sono tutte dei form-check di tipo

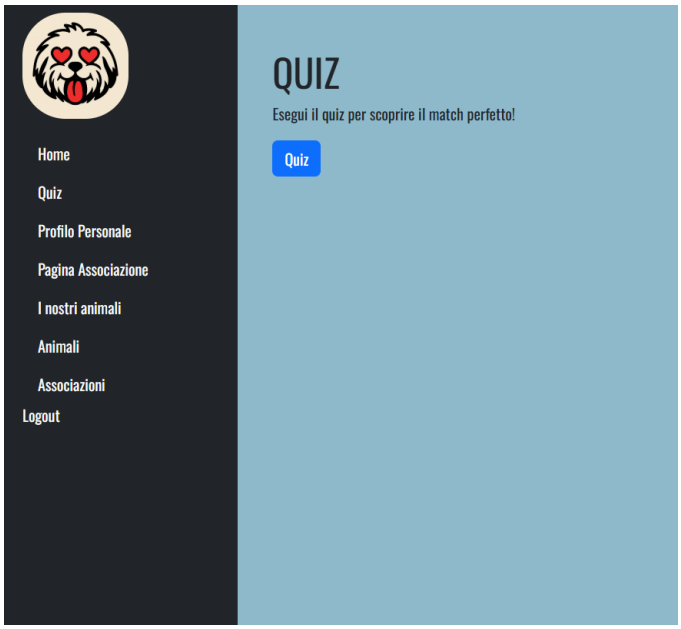


Figura 5.12: Apertura della pagina "Quiz".

radio, che mostrano alcune opzioni (tra le due e le cinque), e permettono di selezionare un'unica risposta. Una volta selezionata la risposta si preme il pulsante apposito per proseguire con la domanda successiva. Nel caso in cui si provi a procedere senza aver selezionato nessuna risposta, la pagina mostra un pop-up di tipo alert che mostra "Seleziona prima una risposta!" e rimane sulla domanda corrente senza proseguire. Al termine delle domande, premendo il bottone per procedere la pagina calcolerà e salverà i relativi valori delle caratteristiche e mostrerà il pop-up "Quiz completato con successo!".

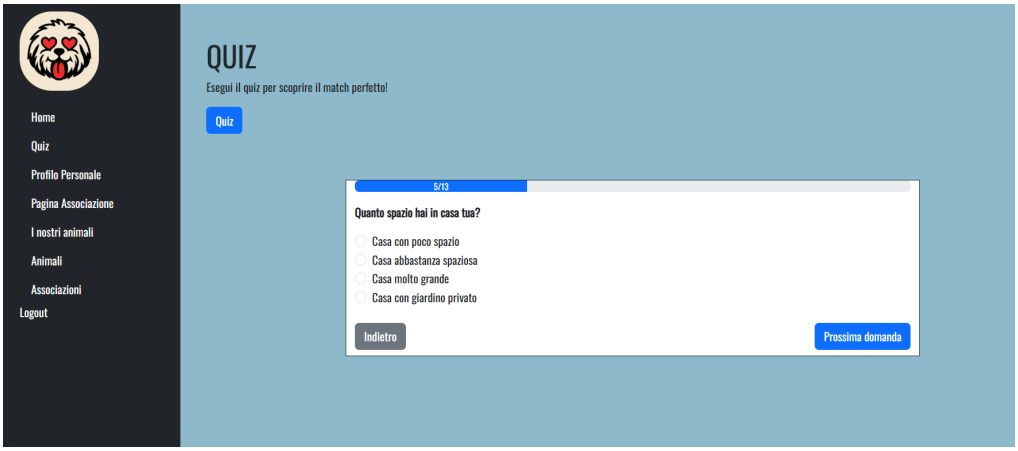


Figura 5.13: Schermata rappresentante lo scorrimento delle domande del quiz.

Il quiz può essere ripetuto quante volte si desidera, accedendo sempre dalla stessa pagina. Ogni volta che viene completato il quiz, vengono calcolati i valori delle caratteristiche per quell'utente, il ché avvia un signals che permette di calcolare anche i valori delle affinità con gli animali. Quando viene ripetuto il quiz, i nuovi valori vengono sovrascritti prendendo il posto dei precedenti, ed il signals avvia il processo di calcolo delle affinità con i valori aggiornati.

5.4 Caratteristiche

Le caratteristiche analizzate sono 11 e sono state immese a sistema (e non modificabili) nel tentativo di rispecchiare tutti quegli elementi che possono influenzare o creare ostacoli nel rapporto tra essere umano ed animale. Le caratteristiche in questione, con le relative descrizioni, sono le seguenti:

- **Movimento - Stimoli**: bisogno dell'animale di giocare e fare movimento.
- **Stare all'aperto - Ambiente**: bisogno dell'animale di stare all'aria aperta.
- **Cure (tempo)**: bisogno dell'animale di qualcuno che si occupi di lui a livello di tempo da dedicargli.
- **Rumore**: quanto l'animale è rumoroso (ad esempio se abbaia tanto o senza motivo).
- **Socialità**: quanto l'animale è socievole.
- **Addestrabilità**: predisposizione dell'animale ad essere addestrato.
- **Compatibilità con bambini**: quanto l'animale va d'accordo con i bambini.
- **Compatibilità con animali**: compatibilità dell'animale con eventuali altri animali (ad esempio cani, gatti ecc).
- **Stato di salute**: se l'animale ha delle malattie e quanto gravi.
- **Problemi comportamentali**: se l'animale ha problemi comportamentali e quanto sono invalidanti.
- **Cure economiche**: importanza a livello economico delle spese da sostenere per le cure dell'animale.

5.4.1 Caratteristiche Animali

Un alto valore della caratteristica "Movimento - Stimoli" indica un animale che ha molto bisogno di fare attività motoria, quindi l'algoritmo dovrebbe cercare di assegnarlo ad un utente che abbia possibilità e predisposizione ad assecondare questo bisogno. Similmente un alto valore della caratteristica "Stare all'aperto - Ambiente" indica la necessità di un'abitazione spaziosa e/o di giardini e spazi esterni facilmente accessibili, e tendenzialmente è pensata per rispecchiare anche le dimensioni dell'animale, poiché un grande animale potrebbe soffrire in piccoli spazi o ad esempio in un monolocale, senza contare che potrebbe anche risultare "ingombrante" per la persona che ci vive, e questo ovviamente rischia da un lato di nuocere direttamente sulla salute dell'animale, e dall'altro di incrinare il rapporto tra i due.

Con "Cure (tempo)" si vuole verificare che la persona a cui l'animale sarà affidato potrà garantirgli (in termini di tempo appunto) il supporto di cui necessita, e di conseguenza una persona che lavora molte ore al giorno ci si aspetta abbia un valore di questa caratteristica minore ad esempio di una persona in pensione. Il "Rumore" è stato inserito per cercare di evitare l'insorgere di problemi futuri che possano condurre addirittura una persona a riportare un animale in un'associazione: un suo alto valore indica ad esempio un animale che abbaia molto, e che quindi necessita sia di pazienza da parte del suo "amico umano", ma soprattutto di un ambiente, magari isolato o senza troppi vicini pronti a lamentarsi, in cui la sua permanenza non venga messa a rischio solo per questo motivo. Con "Socialità" si vuole verificare la predisposizione dell'animale a fare "amicizia" e quindi un basso valore in questo caso potrebbe indicare o la necessità di essere educato a stare in determinati contesti oppure che sarebbe indicato per persone introverse che vogliono un animale che li rispecchi il più possibile.

L'"Addestrabilità" è legata a molte delle altre caratteristiche, in quanto può racchiudere il rumore, la socialità, eventuali problemi comportamentali, o semplicemente insegnarli a seguire certe indicazioni da parte dell'uomo: un suo alto valore indica un'alta predisposizione all'addestramento. La "Compatibilità con bambini" e quella "con animali" è essenziale per evitare conflitti con altri elementi già presenti nell'ambiente in cui l'animale in questione andrà a vivere, che potrebbero costituire un grande ostacolo e portare alla "restituzione" (abbandono) dell'animale all'associazione (ovvero proprio ciò che questa piattaforma cerca di evitare). Lo "Stato di salute", insieme alle "Cure economiche" e alla precedente caratteristica "Cure (tempo)", cerca di portare in luce, al meglio delle possibilità, delle necessità dell'animale che non possono essere trascurate, possono richiedere una grande spesa sia in termini economici che di tempo, e di conseguenza un loro alto valore (in tutti e tre i casi), indica che l'animale dovrebbe essere assegnato a qualcuno che: possibilmente abbia già un po' di esperienza, abbia l'adeguato tempo a disposizione,

e sicuramente abbia abbastanza risorse economiche per garantire all'animale ciò di cui ha bisogno. Infine i "Problemi comportamentali" servono, insieme all'addestrabilità ed al rumore, a fare un match adeguato anche sulla base del "carattere" dell'animale e della persona e quindi non creare inutili futuri conflitti tra i due (per quanto possibile).

Come mostrato in Figura 5.14, tutte queste caratteristiche, per quanto riguarda gli animali, vengono inserite a database, tramite la piattaforma e la pagina "i nostri animali" direttamente dall'associazione che ha in carico l'animale in questione ed ovviamente possono essere modificate nel tempo, a seconda di come cresce e delle esigenze e caratteristiche che l'animale può sviluppare nel tempo. Ciascuna di queste caratteristiche, del form relativo all'animale, può avere un valore compreso tra 0 e 10, dove 0 indica sostanzialmente l'assenza di quella caratteristica in quell'animale, mentre il 10 è la sua massima espressione: ad esempio "Rumore" pari a 0 indica un animale che non abbaia praticamente mai e che non si sente nemmeno, mentre "Socialità" pari a 10 spetta ad un animale che addirittura soffrirebbe a non avere contatti e relazioni esterne al di fuori del suo "genitore umano". Queste caratteristiche infatti non hanno una connotazione negativa o positiva, sono semplicemente degli indicatori con cui si cerca di riassumere e descrivere al meglio il carattere dell'animale per potergli offrire ciò di cui ha bisogno, anche se purtroppo, e proprio perché, alla fine non spetterà a lui la scelta di "con chi andare a vivere" e l'essere umano troppo spesso si rivela egoista.

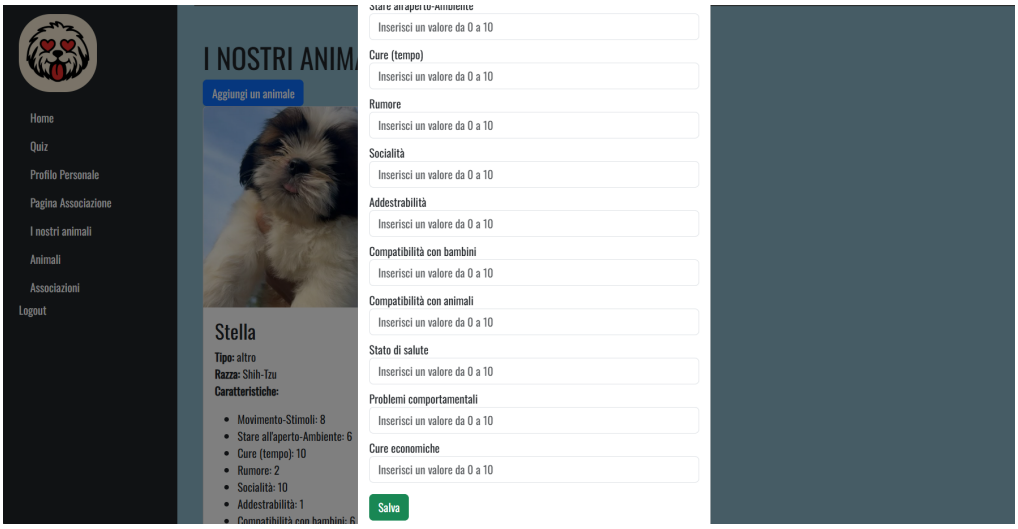


Figura 5.14: Schermata rappresentante l'inserimento di un'animale sul sito da parte dell'associazione.

5.4.2 Caratteristiche Utenti

Dal lato utente, queste caratteristiche non vengono direttamente immesse a sistema ma, per cercare di ricavarle nel modo più oggettivo possibile, vengono calcolate per ogni utente sulla base delle risposte che vengono date durante il quiz apposito. Le domande di questo quiz sono 13, quasi ogni domanda associata ad una particolare caratteristica, tranne per dei casi in cui una domanda può rivelare aspetti di più di una caratteristica. Le domande sono le seguenti:

1. Quanto tempo hai per assistere un animale domestico?
2. Quanta disponibilità economica hai per prenderti cura di un animale domestico?
3. Quanta predisposizione hai per far fare attività al tuo animale domestico?
4. Quanto tempo passi già, o saresti disposto a passare, all'aperto insieme al tuo animale domestico?
5. Quanto spazio hai in casa tua?
6. Hai vicini o vivi in un condominio in cui potrebbe dare fastidio il rumore?
7. Quanto saresti predisposto a portare il tuo futuro animale ad addestrarlo? (Per eventuali problemi comportamentali o rumore)
8. Quanto vorresti che fosse socievole il tuo animale domestico?
9. Quanto vuoi che il tuo animale domestico sia predisposto all'addestramento cinofilo?
10. Quanto è necessario che il tuo futuro animale sia compatibile con dei bambini?
11. Quanto è necessario che il tuo futuro animale domestico sia compatibile con altri animali?
12. Sei disposto a prenderti cura di un animale già malato?
13. Sei disposto ad adottare un animale con problemi comportamentali?

Ognuna di queste domande offre la possibilità di una sola risposta tramite una selezione di opzioni disponibili, il cui numero può variare da domanda a domanda (da un minimo di due risposte ad un massimo di 5). Ogni domanda si riferisce in maniera diretta ad almeno una caratteristica: ad esempio le domande numero

4 e 5 si riferiscono entrambe solo alla caratteristica "Stare all'aperto - Ambiente", mentre la domanda 7 valuta sia "Rumore" che "Problemi comportamentali".

Una volta completato il quiz da parte dell'utente, viene avviato il blocco Javascript che permette il calcolo delle caratteristiche. In particolare questo script mappa per prima cosa le risposte ai punteggi delle caratteristiche, come mostrato dal codice in Figura 5.15, assegnando a ciascuna dei valori che possano, al termine, risultare interni ad un range compreso tra 1 e 10 e dividendo quindi questo spazio di valori in "sotto-sezioni" rappresentative a seconda del numero di domande che li influenzano. Infatti per le caratteristiche che sono influenzate solo da una domanda, come ad esempio nel caso di "Movimento - Stimoli" influenzata unicamente dalla domanda numero 3, si è cercato di dividere il range (1, 10) in modo significativo per il numero di risposte che la domanda offre, in questo caso infatti la domanda propone 5 differenti risposte, ed i valori assegnati a quella caratteristica a seconda della scelta sono 1, 4, 6, 8 o 10. Per raggiungere lo stesso obiettivo, le caratteristiche messe in luce da più di una domanda si dividono lo spazio già tra le domande, questo è il caso della caratteristica "Rumore" influenzata dalle domande 6 e 7, e che quindi si dividono un range di 5 punti ciascuna: nella domanda numero 6, che offre solo due opzioni di scelta, la caratteristica può assumere i valori 1 oppure 5, mentre nella domanda 7 con tre opzioni, i valori (che vengono sommati ai precedenti) possono essere 1, 2 oppure 5, per raggiungere così un range che va da 2 a 10.

Sempre all'interno di questo script, la funzione "prossimo()", legata al bottone "Prossima domanda" tramite la proprietà "onclick", oltre a verificare la compilazione della domanda e a permettere di proseguire con quella successiva, si occupa anche di salvare le risposte date e verificare che la domanda attuale non sia l'ultima tramite una condizione di "if", altrimenti (una volta raggiunta la fine del quiz) calcola i punteggi richiamando la funzione computeScores(answers, pointsMap) e passandogli come parametri il vettore di risposte e la mappa iniziale. Infine invia i risultati al server richiamando la funzione saveResults(scores).

La funzione computeScores(answers, pointsMap) restituisce un oggetto con i totali aggregati. La funzione saveResults(scores) è una funzione asincrona che invia i risultati al server tramite fetch, cioè un'API per fare richieste HTTP (in questo caso di tipo POST).

Al completamento del quiz, l'utente, tramite la pagina "Animali", potrà vedere in tempo reale le affinità calcolate sulla base delle sue caratteristiche.

```
const pointsMap = {
  1: { //Cure(tempo):tempo hai per assistere un animale
    "1": { "Cure (tempo)": 1},
    "2": { "Cure (tempo)": 6},
    "3": { "Cure (tempo)": 10},
  },
  2: { //disponibilità economica
    "1": { "Cure economiche": 1},
    "2": { "Cure economiche": 6},
    "3": { "Cure economiche": 10},
  },
  3: { //far fare attività al tuo animale
    "1": { "Movimento-Stimoli": 1 },
    "2": { "Movimento-Stimoli": 4 },
    "3": { "Movimento-Stimoli": 6 },
    "4": { "Movimento-Stimoli": 8 },
    "5": { "Movimento-Stimoli": 10 },
  },
  4: { //passare, all'aperto insieme al tuo animale: Stare all'aperto-Ambiente
    "1": { "Stare all'aperto-Ambiente": 1 },
    "2": { "Stare all'aperto-Ambiente": 3 },
    "3": { "Stare all'aperto-Ambiente": 5 },
  },
  5: { // Stare all'aperto-Ambiente
    "1": { "Stare all'aperto-Ambiente": 1 },
    "2": { "Stare all'aperto-Ambiente": 3 },
    "3": { "Stare all'aperto-Ambiente": 4 },
    "4": { "Stare all'aperto-Ambiente": 5 },
  },
  6: { //potrebbe dare fastidio il rumore? 1-Si; 2-No
    "1": { "Rumore": 1 },
    "2": { "Rumore": 5 },
  },
  7: { // Rumore+Problemi comportamentali
    "1": { "Rumore": 1, "Problemi comportamentali": 1 },
    "2": { "Rumore": 2, "Problemi comportamentali": 3 },
    "3": { "Rumore": 5, "Problemi comportamentali": 5 },
  },
  8: { //Socialità
    "1": { "Socialità": 1 },
    "2": { "Socialità": 6 },
    "3": { "Socialità": 10 },
  },
  9: { //Addestrabilità
    "1": { "Addestrabilità": 1 },
    "2": { "Addestrabilità": 6 },
    "3": { "Addestrabilità": 10 },
  },
  10: { //Compatibilità bambini
    "1": { "Compatibilità con bambini": 1 },
    "2": { "Compatibilità con bambini": 6 },
    "3": { "Compatibilità con bambini": 10 },
  },
  11: { //Compatibilità animali
    "1": { "Compatibilità con animali": 1 },
    "2": { "Compatibilità con animali": 6 },
    "3": { "Compatibilità con animali": 10 },
  },
  12: { //Stato di salute
    "1": { "Stato di salute": 1 },
    "2": { "Stato di salute": 6 },
    "3": { "Stato di salute": 10 },
  },
  13: { //Problemi comport.
    "1": { "Problemi comportamentali": 1 },
    "2": { "Problemi comportamentali": 2 },
    "3": { "Problemi comportamentali": 5 },
  },
};
```

Figura 5.15: Codice della mappa per il calcolo delle carattristiche-utente.

Capitolo 6

Analisi dei risultati

In questo capitolo si analizzano i valori sia di input che di output: a partire dalle caratteristiche di animali e utenti vengono analizzate le affinità tra di essi, per capire quale metrica sia più affidabile nel progetto in questione.

6.1 Dati in input

Per poter analizzare i risultati da punti di vista differenti, per questa analisi si sono scelti degli utenti randomici con caratteristiche tra loro differenti, per coprire il maggior numero possibile di sfaccettature. Similmente è stato fatto per gli animali.

6.1.1 Utenti

Il primo utente analizzato, Utente Uno, è uno studente universitario con molto tempo libero. Vive da solo in un bilocale di un condominio ed ha un piccolo balconcino. Non lavora, è a carico dei genitori, che possono però garantirgli tutto ciò di cui necessita. Questo utente ama particolarmente i cani. Non avrebbe problemi a portare il suo piccolo amico in centri perché venga addestrato, ma data la sua abitazione sarebbe un problema se abbaiasse troppo infastidendo i vicini. Lui ha uno stile di vita socievole e vorrebbe che il suo fedele amico a quattro zampe lo rispecchiasse, e soprattutto che andasse d'accordo con il cane della sua ragazza.

Il secondo utente, Utente Due, è un signore in pensione, che va verso gli 80 anni. Vive in una piccola villetta indipendente e un po' isolata con un ampio

giardino. Questo utente è sordo e data anche la posizione della sua abitazione non sarebbe assolutamente un problema il rumore. Però a causa dell'età e dei vari problemi di salute non potrebbe prendersi adeguatamente cura di un piccolo amico con particolari necessità di stimoli o di movimento. Vorrebbe in particolare un cane per avere compagnia e sentirsi più al sicuro nella propria abitazione, abitazione che non è frequentata nè da bambini nè da animali, e in generale non vede spesso ospiti che vengano a fargli visita, quindi non è necessario che il piccolo amico a quattro zampe sia socievole. Ha una grossa disponibilità economica per cui non sarebbe un problema prendersi cura di un animale con un ingente bisogno economico di cure, ma non ha intenzione di portarlo ad addestrare.

Il terzo utente, Utente Tre, è una madre di famiglia con tre bambini sotto i 12 anni e che lavora in ospedale. Vorrebbe un animale domestico per educare i suoi bambini a prendersene cura e farli crescere con una compagnia fedele come solo un amico a quattro zampe sa dare. Questa famiglia vive in un appartamento spazioso all'interno di un bel condominio, nel cui ingresso vi è anche un piccolo giardino comune. Non hanno altri animali. Data la sua vita frenetica non avrebbe tempo per portare il nuovo componente della famiglia in un centro per l'addestramento o di garantirgli adeguata attenzione nel caso di problemi di salute o comportamentali che necessitino di una supervisione.

Date le loro predisposizioni, le caratteristiche di ognuno dei tre utenti ottenute tramite il quiz sono riportate in Tabella 6.1.

Caratteristiche	Utente 1	Utente 2	Utente 3
Movimento-Stimoli	10	1	8
Stare all'aperto-Ambiente	6	10	7
Cure(tempo)	10	10	1
Rumore	3	7	2
Socialità	10	1	10
Addestrabilità	10	1	1
Compatibilità con bambini	6	1	10
Compatibilità con animali	10	1	6
Stato di salute	1	10	6
Problemi comportamentali	5	8	2
Cure economiche	6	10	6

Tabella 6.1: Valori Caratteristiche-Utenti

6.1.2 Animali

Il primo animale si chiama Ranny, è un cane, in particolare un volpino di piccole dimensioni che non necessita di ambienti particolarmente spaziosi. È un cagnolino vivace, molto socievole e va d'accordo sia con altri animali che con i bambini. Non presenta problemi di salute o comportamentali e non abbaia eccessivamente. Ha bisogno di fare movimento e di avere molti stimoli ed è predisposto ad essere addestrato.

Il secondo animale è un cane di nome Red, è un pastore tedesco. È di grandi dimensioni, necessita di molto spazio ed è anziano. Ha alcuni problemi di salute e necessita di cure economicamente dispendiose. È molto diffidente e abbaia molto, ma non presenta significativi problemi comportamentali. Non è particolarmente predisposto all'addestramento cinofilo.

Il terzo animale è una gattina di nome Taissa, è di razza europea e di colore grigio. Si tratta di un animale molto giocherellone e socievole, che prende subito confidenza ed ama farsi coccolare per ore. È molto compatibile sia con i bambini che con altri animali e non presenta nè problemi di salute nè comportamentali.

Il quarto animale è un cane, un labrador, di nome Perla. Lei è di grandi dimensioni, molto giocherellona, ha bisogno di ampi spazi e possibilmente un giardino, ma soprattutto di molti stimoli e di fare movimento. Non presenta problemi di salute, però abbaia (non eccessivamente) ed ha problemi comportamentali che si manifestano principalmente con estranei e in luoghi aperti, questo anche perché si lega facilmente e diventa protettiva con la sua famiglia. È predisposta all'addestramento cinofilo ed è molto compatibile con i bambini, non altrettanto con gli altri animali.

Il quinto animale è un cane, un pastore australiano, di nome Spark. Necessita di ampi spazi, ama stare all'aperto e fare movimento, è molto socievole e compatibile soprattutto con altri animali. Non presenta problemi comportamentali particolari ma abbaia molto. Non è facilemnte addestrabile. Non ha bisogno di particolari cure.

Dato il quadro generale degli animali, i valori precisi assegnati dall'associazione per ogni caratteristica sono riportati in Tabella 6.2.

Caratteristiche	Ranny	Red	Taissa	Perla	Spark
Movimento-Stimoli	10	2	8	9	8
Stare all'aperto-Ambiente	6	10	7	10	9
Cure(tempo)	4	9	2	2	2
Rumore	3	9	1	6	7
Socialità	10	2	9	6	9
Addestrabilità	8	1	3	8	4
Compatibilità con bambini	8	2	9	7	7
Compatibilità con animali	8	1	7	3	8
Stato di salute	1	8	3	1	1
Problemi comportamentali	4	5	1	8	4
Cure economiche	4	9	3	2	2

Tabella 6.2: Valori Caratteristiche-Animali

6.1.3 Situazione complessiva in input

Già ad un livello superficiale si possono avanzare le prime ipotesi di match, che, per come sono stati appositamente creati questi utenti, dovranno essere rispecchiate dagli output e dalle metriche successivamente.

Infatti per l'Utente Uno è stato creato, in maniera fittizia, appositamente il volpino di nome Ranny, e ci si aspetta dunque che l'affinità maggiore sia riscontrata proprio per questo match. Red ci si aspetta risulti poco affine, poiché già dalla descrizione non rispecchia quasi nessuna caratteristica dell'utente. Con Taissa potrebbe esserci una certa affinità, anche se l'utente predilige i cani. Per quanto riguarda Perla gli ostacoli di un possibile rapporto sarebbero costituiti da: gli ampi spazi di cui l'animale ha bisogno, in conflitto con il bilocale in cui vive l'utente, ma mitigati dalla sua predisposizione a passare molto tempo all'aperto; il rumore di un cane che abbaia in un condominio; ed una compatibilità maggiore con i bambini che con gli altri animali. Infine, per quanto riguarda Spark, ci si aspetta una possibile compatibilità "moderata", in quanto mentre certi aspetti sembrano essere perfettamente rispecchiati, altri parametri possono costituire un forte ostacolo (come ad esempio il rumore e la necessità di spazi ampi).

Similmente l'Utente Due è stato immaginato, e creato appositamente, maggiormente affine al pastore tedesco Red. Ranny e Taissa, principalmente per la socievolezza di cui sono caratterizzati e le attenzioni (in termini di stimoli e movimento) di cui hanno bisogno, ci si aspetta che costituiscano i match meno compatibili con questo utente. Perla e Spark potrebbero essere delle buone soluzioni, se non fosse,

anche in questo caso, principalmente per gli stimoli di cui hanno bisogno e per la socialità che li caratterizza.

Infine l'Utente Tre dovrebbe risultare maggiormente affine a Taissa, e successivamente a Ranny e Spark. Mentre per quanto riguarda Red la compatibilità dovrebbe risultare la più bassa. Perla costituisce un punto intermedio con caratteristiche affini e altre meno.

6.2 Output

Data la progettazione della piattaforma, per ogni coppia utente-animale sono calcolate cinque differenti metriche di affinità. L'obiettivo di questa analisi è di comprendere i punti di forza e di debolezza di queste metriche, per raggiungere la consapevolezza di quale sia la più efficace per il caso d'uso in questione.

6.2.1 Utente Uno

Per l'Utente Uno, le affinità restituite dalla piattaforma, per ogni animale e per ogni metrica, sono riportate in Tabella 6.3. I valori in tabella sono tutti in percentuale.

Metriche	Ranny	Red	Taissa	Perla	Spark
Euclidea	78,05	38,43	60,69	58,99	62,83
Coseno	96,24	62,74	86,62	84,40	87,58
Pearson	89,28	21,01	71,40	59,38	66,99
Jaccard	81,82	36,36	72,73	54,55	63,64
M.S.D.	95,18	62,09	84,55	83,18	86,18

Tabella 6.3: Affinità relative all'Utente Uno

Dalla Tabella 6.3 risulta evidente la forte affinità tra l'Utente Uno e Ranny, che era il legame desiderato, ma ogni metrica mette in luce affinità differenti. Per ognuna delle metriche il legame più forte è con Ranny, mentre quello meno consigliato è con Red.

La distanza euclidea, la similarità del coseno e la differenza quadratica media ordinano gli animali, per questo utente, nel seguente modo:

Ranny > Spark > Taissa > Perla > Red.

Il coefficiente di Pearson e l'indice di Jaccard li ordinano come segue:

Ranny > Taissa > Spark > Perla > Red.

In linea generale tutte e cinque le metriche rispecchiano le aspettative per quanto riguarda l'ordinamento anche se la differenza quadratica media, ad esempio, come la similarità del coseno, restano su valori elevati, in particolare al di sopra del 60 %, mostrando piccole differenze, e questo potrebbe non spingere l'utente ad una scelta pienamente intelligente e consapevole; mentre la distanza euclidea, il coefficiente di Pearson e l'indice di Jaccard, che raggiungono rispettivamente anche valori di 34,43 % 21,01 % e 36,36 %, possono suggerire affinità migliori.

6.2.2 Utente Due

Per quanto riguarda l'Utente Due, sono riportati in Tabella 6.4 i valori delle sue affinità, per ogni metrica e con ogni animale, e come per il caso precedente, sempre da considerarsi in percentuale.

Metriche	Ranny	Red	Taissa	Perla	Spark
Euclidea	31,91	85,86	34,43	40,61	37,26
Coseno	49,23	97,99	46,74	59,99	54,52
Pearson	6,88	97,29	18,36	32,06	21,22
Jaccard	9,09	90,91	18,18	36,36	27,27
M.S.D.	53,64	98,00	57,00	64,73	60,64

Tabella 6.4: Affinità relative all'Utente Due

Dalla Tabella 6.4 viene rispettata l'aspettativa di ottenere il legame maggiore con Red e minore con Ranny (per quasi tutte le metriche) o al massimo con Taissa (per la similarità del coseno).

L'odrinamento, secondo la distanza euclidea, il coefficiente di Pearson, l'indice di Jaccard e la Mean Squared Difference è il seguente:

Red > Perla > Spark > Taissa > Ranny

Per quanto riguarda la similarità del coseno, l'ordine resta simile ma è:

Red > Perla > Spark > Ranny > Taissa

Una sola metrica quindi sembra discostarsi, anche se non in maniera rilevante poiché sia Ranny che Taissa sono da considerarsi le affinità minori. È sicuramente più rilevante come, anche in questo caso, le varie metriche oscillino tra range differenti. Anche se in questo caso, diversamente da quello precedente, tutte e cinque le metriche scendono con valori anche inferiori ad un'ipotetica soglia di 60 %, permettendo quindi di avere un immediato riscontro per quanto riguarda animali totlamente incompatibili a loro. La similarità del coseno e la Differenza quadratica media, in questo caso, scendono anche sotto il 60%, permettendo all'utente di identificare legami che non rispecchiano delle affinità. La distanza euclidea addirittura presenta un solo caso, quello relativo a Red, come affine, mentre tutti gli altri risultano inferiori del 50 %. Il coefficiente di Pearson e l'indice di Jaccard sembrano essere anche in questo caso le metriche con una maggior oscillazione, arrivando a sconsigliare fortemente certi match (addirittura del 6,88% o del 9,09%).

6.2.3 Utente Tre

Per l'Utente Tre, i valori in percentuale delle sue affinità sono riportati in Tabella 6.5.

Metriche	Ranny	Red	Taissa	Perla	Spark
Euclidea	69,10	44,98	84,05	58,77	70,15
Coseno	88,71	62,04	96,87	78,74	88,60
Pearson	75,61	34,58	94,36	56,33	76,40
Jaccard	72,73	45,45	81,82	45,45	72,73
M.S.D.	90,45	69,73	97,45	83,00	91,09

Tabella 6.5: Affinità relative all'Utente Tre

Dalle affinità calcolate è evidente come il match più forte risulti essere con Taissa mentre il peggiore con Red (in un caso, quello dell'indice di Jaccard, a parimerito con Perla).

L'ordine delle affinità secondo la distanza euclidea, il coefficiente di Pearson e la mean squared difference è il seguente:

Taissa > Spark > Ranny > Perla > Red

Per la similarità del coseno l'ordine cambia, ma non in maniera significativa:

Taissa > Ranny > Spark > Perla > Red

Con l'indice di Jaccard l'ordine diventa:

Taissa > Ranny = Spark > Red = Perla

Nonostante qualche piccola differenza, l'ordinamento risulta efficiente in tutti e cinque i casi. Anche in questo contesto però, la similarità del coseno e la differenza quadratica media riportano per tutti valori al di sopra di un'ipotetica soglia del 60 %, non enfatizzando così le differenze tra gli animali e quindi rischiando di guidare l'utente verso una decisione poco consapevole. La distanza euclidea, il coefficiente di Pearson e l'indice di Jaccard, risultano invece, come nei casi precedenti, delle buone guide per prendere una decisione, permettendo di visulaizzare affinità minori al 60 % per animali poco compatibili.

6.3 Analisi delle metriche

In generale, come si può osservare dai dati ottenuti, tutte le metriche hanno ordinato in maniera logica i vari match. Ogni metrica però contiene un significato diverso e riporta valori, che possono essere più o meno significativi, su dei range differenti di volta in volta.

6.3.1 Distanza Euclidea

La Distanza Euclidea risulta essere un'ottima metrica per questo progetto. Essa infatti, calcolando l'effettiva distanza tra le caratteristiche dell'utente e dell'animale, propone non solo un ordinamento adeguato, ma riesce anche a proporsi come una guida nella scelta dell'utente, mettendo in luce quei casi in cui vi è una sostanziale incompatibilità e rispecchiandoli con una percentuale inferiore al 60 % o addirittura al 50 %. Negli esempi proposti, questa metrica oscilla in un range compreso circa tra 30 % ed 85 %, permettendo quindi sia di individuare i casi "non affini", sia quelli che invece lo sono. Trattandosi di una misura di distanza, risponde in modo proporzionale alle differenze numeriche delle caratteristiche tra utente e animale ed i valori che propone sono facilmente interpretabili ed intuitivi, infatti più i valori delle caratteristiche coincidono o si avvicinano, e maggiore sarà l'affinità risultante. Il fatto che si muova su un range compreso tra 30-85% può inoltre costituire un vantaggio per l'utilizzatore finale, che può percepire il match come più realistico,

soprattutto rispetto a metriche che propongono più facilmente valori tendenti ai limiti (quindi verso lo 0% o il 100%): questa metrica per l'analisi effettuata può risultare maggiormente "sincera", non offrendo compatibilità "perfette" o totalmente inesistenti ma restando su differenze reali, evidenziate dai suoi valori. Inoltre questa metrica è in grado di evidenziare anche compatibilità "parziali", nei casi in cui il suo valore oscilla tra il 50-65%, come nel caso dell'Utente Uno con Taissa o Perla, con cui presenta alcuni punti in comune.

Calcolando principalmente la differenza di punteggi tra due vettori, questa metrica può risultare molto sensibile anche a singoli scostamenti, ad esempio per profili simili ma con due caratteristiche agli estremi opposti. Inoltre non considera eventuali "pattern", come un utente ed un animale che hanno punteggi alti sulle stesse caratteristiche e bassi su altre, sempre in comune, ma calcola appunto la differenza punto per punto.

Il significato di questa metrica, anche per come è stata implementata all'interno del progetto, è di mettere in luce l'effettiva distanza (poi convertita in affinità e quindi "vicinanza") tra le varie coppie di punti utente-animale, caratteristica per caratteristica, accettuandone la distanza tramite l'elevazione al quadrato delle differenze: l'utente ha un vettore di caratteristiche, l'animale ne ha un altro, e questa metrica si propone di constatare quanto questi due siano vicini nello spazio. La sua implementazione nel progetto ha fornito risultati rilevanti e significativi, come mostrato dagli esempi proposti.

6.3.2 Cosine Similarity

La Similarità del Coseno misura l'angolo tra due vettori, cercando quindi di mettere in luce più che altro l'andamento, o il pattern, tra l'utente e l'animale. Questo focus sul pattern permette di evidenziare quelle che possono essere caratteristiche importanti comuni ad entrambi, come nel caso dell'Utente Due con Red, che hanno effettivamente valori elevati delle stesse caratteristiche, come "Stare all'aperto-Ambiente", "Cure (tempo)" o lo "Stato di salute", ed entrambi valori bassi di altre caratteristiche come "Movimento-Stimoli" o "Socialità".

La ricerca di pattern che caratterizza questa metrica però risulta anche un punto debole per questo progetto, in quanto un utente con caratteristiche tutte pari ad 1 potrebbe ottenere un'affinità perfetta (100%) con un animale con caratteristiche tutte pari a 10, o similmente un utente (1, 2, 3, 1, 2, 3 ...) otterebbe un match migliore con un animale (2, 6, 9, 2, 9 ...) che con un animale (2, 1, 2, 1, 2, 1 ...), in quanto verrebbero riscontrati dei pattern che farebbero esplodere la metrica senza analizzare la distanza effettiva tra i valori.

Inoltre, poiché la scala dei valori delle caratteristiche è compresa tra [0, 1], ed in particolare vi sono spesso valori medio-alti, il prodotto scalare dei vettori non solo non può mai essere negativo (il range è costituito da valori tutti positivi) ma risulta anche elevato, e similmente anche le norme dei vettori risulteranno elevate, portando a valori di questa metrica vicini ad uno, e quindi ad affinità tendenti al 100 % o comunque molto più alte di altre metriche. Questo è il caso ad esempio del match tra Utente Uno e Ranny, pari a 96,24 %, ma sempre con questo utente lo si può riscontrare analizzando l'affinità con gli altri animali e vedendo come essa non scenda sotto il 62,74 % (anche per l'Utente Tre presenta solo valori superiori al 62 %). Questa caratteristica porta la metrica a non essere estremamente realistica, a differenza della distanza euclidea ad esempio, e a non fornire una buona e consapevole guida per la scelta dell'utente.

6.3.3 Pearson Correlation Coefficient

Il Coefficiente di Correlazione di Pearson misura in parole povere l'andamento dei valori e la correlazione tra essi: non solo quando due caratteristiche si muovono insieme, ma anche quanto "forte" sia questo legame.

Nel caso dell'Utente 1 con Ranny ad esempio, un'affinità dell'89,28 % indica una forte correlazione, cioè non solo una gerarchia simile tra le caratteristiche, ma anche lo stesso pattern, e similmente accade per l'Utente Due con Red, con un'affinità pari a 97,29 %. Valori bassi come quelli tra Utente 1 e Red (21,01%) o Utente 2 e Ranny (6,88%) indicano profili concettualmente diversi o addirittura opposti, come accade proprio in questi due casi: questi due utenti infatti sono stati creati appositamente il più distanti possibile l'uno dall'altro, e mentre Ranny rispecchia il primo, Red deve rispecchiare il secondo; di conseguenza questi due match dovrebbero risultare (come accade) i meno indicati già per costruzione, e vogliono rispecchiare il caso di un utente ed un animale che non convergono praticamente su nessun aspetto.

Per molti aspetti questa metrica può essere paragonata alla similarità del coseno: infatti entrambe misurano l'affinità tra due vettori, ricercano pattern e sono sensibili alla direzione verso cui puntano le "preferenze" dell'utente (e dell'animale). Pearson però è sensibile alle differenze rispetto alla media, così da mettere maggiormente in luce incoerenze tra le caratteristiche: infatti questa metrica, più che l'andamento, studia come i valori si discostano dalla propria media, ed è questo l'andamento che prende in considerazione. Quindi per quanto risulti molto efficace, anche per gli esempi forniti precedentemente, rischia anche questa metrica di enfatizzare troppo dei pattern anche in casi in cui i valori assoluti non coincidono. Questo rischio è minore del caso della cosine similarity poiché valuta almeno che l'andamento sia proporzionale rispetto alla propria media, quindi ha una sorta di fase in più. Il

risultato di questa metrica potrebbe inoltre essere distorto anche a causa di un singolo outlier, o meglio di un'unica caratteristica che si discosta molto più delle altre.

In generale però, questa metrica offre risultati dinamici e significativi, soprattutto nel caso preso in analisi, ed è un buon modo per valutare complessivamente l'affinità dei profili.

6.3.4 Jaccard Similarity

Per come è stato studiato ed implementato il quiz, in realtà i valori delle caratteristiche degli Utenti non sono tutti possibili: per ogni caratteristica vi sono alcuni valori rappresentativi che l'Utente può "ottenere", in base alle risposte date alle varie domande. In questo modo, si ha in realtà già una perdita di informazioni alla base, poiché dato il range (0, 10), in realtà l'utente può ottenere dei valori delimitati che corrisponderebbero ad una sorta di "sotto-range". Ad esempio per una caratteristica che immagina avere al minimo potrebbe ottenere un 1 che non rappresenta solo se stesso ma il sotto-gruppo 0-3 (come indice di un valore basso). La scelta che ha portato ad una tale semplificazione è stata dettata dal rischio di ottenere valori meno rappresentativi ancora, offrendo all'utente la possibilità di scegliere tra tutti i valori e quindi rischiando che per la troppo vasta gamma di scelta, l'utente non fosse in grado di dare una risposta significativa e che rispecchiasse lo stato attuale e reale, rischiando così una maggiore perdita di informazioni, o meglio l'immissione di informazioni errate e inutili.

La similarità di Jaccard è una metrica ottima per dati binari, in cui conta tipicamente verificare l'assenza o la presenza di un elemento, e si presta invece poco ad un contesto come questo di valori continui (ma anche solo di più di due valori). In questo caso risulta quindi un'ulteriore semplificazione della semplificazione, e quindi un'ulteriore perdita di informazioni, dato che da un piccolo numero di "sotto-gruppi", si arriva ad averne addirittura solo più due: i valori maggiori di 5 vengono contrassegnati con un 1 mentre quelli minori o uguali a quella soglia diventano 0, come se una caratteristica fosse presente solo se l'utente o l'animale la possiedono con un valore elevato. L'aggiustamento che è stato fatto proprio per poter implementare questa metrica infatti corrisponde ad una divisione dei valori in "alto" e "basso" (rispetto alla soglia scelta di 5).

A questo punto la similarità di Jaccard analizza quante caratteristiche, per l'animale e l'utente in questione, sono entrambe alte o entrambe basse rispetto al totale delle caratteristiche presenti.

Nonostante l'enorme semplificazione che è stata implementata, risulta una misura comunque utile per evidenziare subito relazioni affini o agli antipodi, raggiungendo anche valori estremamente bassi per match che risultano effettivamente "opposti", come nel caso dell'Utente Due con Ranny con un'affinità pari a 9,09 %. Permette quindi anche l'immaginazione di una soglia tra il 55-65 % sotto la quale è fortemente sconsigliato il match.

Purtroppo però, la significativa perdita di informazioni e il modo in cui è strutturato non permettono di far emergere differenze tra match "simili" ma diversi, o in generale di cogliere le varie sfumature delle caratteristiche (quella è proprio l'informazione quantitativa che viene persa), e così facendo, come nel caso dell'Utente Tre, alcuni animali possono essere considerati uguali come affinità anche se le loro caratteristiche potrebbero nascondere sfumature differenti e portare a risultati diversi.

Inoltre rischia di essere un grande ostacolo per casi che si aggirano attorno alla soglia: caratteristiche con valori tra il 4 ed il 6 potrebbero ottenere una classificazione differente e risultare incompatibili con altre caratteristiche sempre attorno a quel valore, ma che sono relativamente più alte o più basse, ottenendo così i valori esattamente opposti (e portando quindi ad un match tendente allo 0 % quando paradossalmente potrebbe essere verso il 100 %).

6.3.5 Mean Squared Difference

La differenza quadratica media (msd) corrisponde ad una sorta di "reciproco" dell'affinità, per come essa viene attualmente calcolata, mentre l'RMSE (root mean square error), corrispondente alla radice quadrata dell'msd, indica la media degli scarti (su scala originale, quindi compresa tra 0 e 10). Ad esempio: l'affinità tra l'Utente Uno e Ranny pari a 95,18 % può essere vista anche come la distanza tra questi due elementi di un msd pari a 4,82 (100 - l'affinità), o ancora come una differenza media di circa 2,2 punti nelle 11 caratteristiche (scarto quadratico medio). Viceversa quando si hanno differenze medie elevate, come tra l'Utente Uno e Red, l'affinità risultante sarà ridotta (62,09 %).

Si tratta quindi di una metrica funzionale al progetto, in quanto, analizzando le caratteristiche punto per punto, riesce ad individuare i match con una maggiore affinità e a garantire un adeguato ordinamento degli elementi. Inoltre essa è facilmente interpretabile ed intuitiva, poiché similmente alla distanza euclidea si occupa di calcolare, appunto, una "distanza", dalla quale viene ricavata la corrispondente affinità.

Questa metrica può essere maggiormente paragonata alla distanza euclidea, dato lo scopo e la costruzione delle due, mentre potrebbe "completare" metriche come la cosine similarity o l'indice di Pearson andando ad enfatizzare proprio quegli aspetti di cui queste ultime due non si occupano: mentre Pearson e Cosine ricercano più dei pattern, la msd mette in luce l'effettiva entità delle discrepanze.

La mean squared difference è una metrica tipicamente sensibile agli outlier, in cui quindi pochi dati di "rumore" possono condizionare (e dominare) il risultato finale. Per costruzione della piattaforma stessa, questo non risulta essere un problema eccessivo, in quanto i dati degli animali sono inseriti dalle associazione dopo loro analisi dei casi in questione, mentre per l'utente i punteggi sono addirittura calcolati dal sistema sulla base delle sue risposte, riducendo al minimo possibile il rischio di dati in input non validi o sbagliati: l'utente non deve assegnare un punteggio alla grandezza di casa sua, deve solo scegliere, tra le alternative proposte, la risposta che meglio rispecchia la sua abitazione.

Il range di questa metrica, analizzando le tre tabelle di output, sembra essere compreso tra il 53 % ed il 98 %: in particolare riporta valori tutti positivi tranne per due match entrambi relativi all'Utente Due, in cui comunque non scende al di sotto del 50 %. Rischia quindi di non essere un'ottima guida per le scelte dell'utente e soprattutto di non sconsigliare in maniera adeguata quegli animali che, date le caratteristiche, hanno esigenze nettamente differenti. Un esempio di ciò è dato in particolare dall'Utente Uno e Red, che grazie alla msd ottengono un'affinità del 62,09 % (considerabile "sufficiente") anche se caratterialmente costruiti come elementi agli antipodi e con esigenze fortemente in contrasto. Questa misura quindi non produce lo stesso risultato della distanza euclidea ad esempio, che nello stesso caso sconsiglia fortemente il match scendendo ad un'affinità di 38,43 %. Questo avviene perchè la distanza euclidea, sommando e radicando le differenze, le restituisce come se si trattasse (e così dovrebbe essere) di una grande distanza globale; la mean squared difference invece, prima fa la media delle differenze quadratiche e poi le scala linearmente su 100 (range 0-100), ma in questo modo è come se diluisse l'impatto delle forti differenze su tutte le varie caratteristiche, facendo alla fine risultare gli elementi posti a confronto "non così tanto differenti".

In conclusione quindi, pur essendo una metrica valida, non risulta essere la migliore per il caso in questione e sicuramente non realistica quanto la metrica della distanza euclidea.

6.3.6 Confronto delle metriche

Come mostrato in Tabella 6.6, la migliore metrica per la piattaforma in questione risulta essere la Distanza Euclidea: è la più realistica, mostra effettivamente la "distanza" tra gli elementi analizzati (cioè quello che si desiderava riscontrare), e permette all'utente di prendere decisioni ponderate e attente, sconsigliando con valori decisi i match non idonei. I suoi unici contro risultano essere la mancanza di pesi, che riguarda una scelta di programmazione e non un problema della metrica in sè, e che comunque possono sempre essere implementati se necessario, ma data la vastità di caratteristiche si è deciso di non caricare eccessivamente l'una o l'altra ma far sì che globalmente si completassero; l'altro punto a sfavore risiede nell'eccessiva severità che si può riscontrare con un gran numero di piccole differenze, che possono quindi gravare tanto sommate, risultato che comunque non è da considerarsi eccessivamente negativo perché produrrebbe un'affinità minore di un match con meno differenze, e quindi porta sempre a risultati più realistici. Per questi motivi risulta attualmente la metrica più significativa e attendibile.

La Cosine Similarity ed il coefficiente di Correlazione di Pearson invece, pur offrendo valori generalmente validi, soprattutto per gli esempi analizzati, poiché ricercano trend e pattern, non risultano ugualmente adeguate e perdono l'importante informazione quantitativa che consiste nella magnitudine delle differenze tra i vettori, non permettendo un quadro completo dei risultati desiderati e anzi, rischiando di evidenziare affinità errate in determinati contesti (vettori proporzionali ma lontani). Queste metriche non risultano quindi adatte al progetto in questione.

L'indice di Jaccard, nella sua estrema semplicità, permette di mettere in luce elementi simili per "macro-categorie", e non risulta nemmeno una pessima scelta; però comporta una sostanziale perdita di informazioni e può provocare errori soprattutto per valori limite (vicini alla soglia considerata). Per questi motivi questa metrica, pur non essendo considerata la più pericolosa, può fornire una prima visione generale ma non è da considerarsi la guida attendibile per prendere decisioni su un gran numero di elementi: proprio su tanti elementi infatti rischierebbe di fornire punteggi di affinità simili (se non addirittura identici) anche per casi che di differenze ne presentano molte.

Infine la Mean Squared Difference, che può essere per molti aspetti paragonata alla Distanza Euclidea (favorisce l'informazione quantitativa invece che l'andamento, è di facile interpretazione e calcola una distanza), si discosta proprio da questa per un dettaglio rilevante, dovuto anche alla trasformazione finale in percentuale, che la porta ad appiattire le differenze diluendole sul totale considerato, e questo ha conseguenze dirette anche sulla scala di valori dei risultati ottenuti, che viene compresso verso la parte alta del range complessivo, risultando non una buona guida

da fornire all'utente per mettergli in evidenza animali compatibili rispetto a quelli fortemente sconsigliati.

Metrica	Pro	Contro	Utilità nel progetto
Distanza Euclidea	Penalizza correttamente le differenze; risultati intuitivi; riflette bene la "distanza reale"; offre risultati "realistici".	Mancanza di pesi; può essere severa per molti scarti piccoli.	**Molto utile** – la più affidabile per il progetto.
Cosine Similarity	Evidenzia pattern simili; resistente a offset e intensità differenti.	Non considera le differenze assolute; può sovrastimare affinità con valori proporzionali ma lontani.	**Inadatta**: si presta meglio ad altri contesti.
Pearson Correlation	Rileva coerenze nell'andamento delle variazioni; metrica robusta; evidenzia relazioni di comportamento.	Non misura la distanza reale; può dare alte affinità anche a coppie con valori molto diversi.	**Mediamente utile**: può fornire un altro punto di vista ma non si presta bene al progetto.
Jaccard Index	Semplice da interpretare; mostra la percentuale di tratti realmente condivisi.	Poco preciso; perdita di informazioni; non distingue tra "simile" e "uguale".	**Buona** – offre una visione d'insieme riassuntiva, ma non è la metrica ricercata per questo progetto.
Mean Squared Difference (MSD)	Semplice, stabile e sensibile al rumore; facile da calcolare.	Appiattisce le differenze; penalizza poco le discrepanze forti; compressione dei punteggi (60–95%).	**Inadatta** – troppo permissiva, non fornisce una guida valida per l'utente.

Tabella 6.6: Confronto tra metriche di similarità: pro, contro e utilità nel progetto.

Capitolo 7

Conclusioni

Lo scopo di questo progetto era di ridurre l'abbandono di animali domestici. Non potendo controllare nè educare la società, si è cercato di creare una piattaforma web che mettesse direttamente in comunicazione utenti e animali domestici affini, ponendo questi ultimi al centro dell'analisi, con la finalità di creare rapporti che possano riscontrare meno ostacoli futuri possibili. Il tutto è stato fatto in modo non troppo evidente, per lasciare all'utente l'apparenza di avere il totale controllo, e così cercare di coinvolgere più persone possibili. Infatti, se ad esempio alcune caratteristiche fossero state implementate come una sorta di filtro bloccante ("chi ha una casa minore di una certa metratura non può neanche visualizzare animali di certe dimensioni") si sarebbe rischiato di perdere in partenza una gamma di utenti particolarmente egoisti con il desiderio di particolari razze di animali che per loro natura sarebbero poco indicati. Non ponendo condizioni vincolanti e lasciando all'utente la possibilità di visualizzare tutti i risultati, a prescindere dalla compatibilità, viene lasciata la possibilità, insieme alla speranza, che esso scelga almeno il match "relativamente migliore" tra quelli di suo gusto. Eventualmente si può dare alle associazioni la possibilità di visionare il profilo dell'utente ed i match così da lasciare a loro la scelta finale.

Ovviamente questa piattaforma non si può sostituire all'uomo nè sviluppare intelligenza emotiva lì dove è carente, ma a livello di fattibilità può risultare uno strumento efficace per contenere questo terribile fenomeno e limitarlo almeno in parte. Tutto il resto è un lavoro che spetta alla società e all'essere umano.

7.1 Valore aggiunto

Questa piattaforma è da considerarsi come un primo passo verso la giusta direzione. Rispetto ad altre piattaforme esistenti essa mette particolarmente in evidenza le esigenze dell'animale ponendo esso al centro di tuttto e non classificandolo per tipologia o razza ma analizzando ogni animale come simile solo a sè stesso, esattamente come viene fatto tendenzialmente per gli utenti. In questa piattaforma si cerca di ristabilire un po' l'ordine: l'uomo può godere del privilegio di prendersi cura di un animale domestico, non il contrario. Quindi si deve anche essere all'altezza di ciò ed idonei per farlo. Vi sono molti passi in avanti che possono ancora essere compiuti, ma con questa piattaforma si spera di dare una maggiore spinta alla "giusta direzione". Nessun animale si sognerebbe mai di abbandonare il proprio amico umano in mezzo all'autostrada, quindi non ha senso che avvenga il contrario.

7.2 Sviluppi futuri

Questo progetto rappresenta una sorta di prototipo, o prima applicazione: vi sono le funzionalità base che permettono di raggiungere l'obiettivo, e ulteriori modifiche, come l'aggiunta di eventuali filtri, possono facilmente essere implementate a partire da questo progetto.

In un'ottica più ampia sarebbe però bello e utile se in questa piattaforma venissero implementate anche nuove funzionalità relative sempre alla sensibilizzazione e all'educazione dell'uomo nel prendersi cura di altri esseri viventi: ad esempio potrebbero essere inseriti articoli relativi a questi temi oppure delle attività semiludiche che intrattengano l'utente e lo invitino a rimanere attivo sulla pagina anche se non ha la possibilità di adottare un animale domestico in quel momento, ma che allo stesso tempo, tramite dei "giochi", lo educhino al rispetto e alla cura e che gli insegnino le basi per poter offrire una vita dignitosa ad un futuro amico a quattro zampe, come in una sorta di gamification della piattaforma.

Inoltre potrebbe essere implementato un sistema di messaggistica che permetta uno scambio diretto di informazioni tra utente e associazione, così da permettere anche alle associazioni di avere un'idea, in base all'affinità del match, alle caratteristiche dell'utente e a domande poste direttamente, degli utenti interessati a un certo animale e poter scegliere con queste informazioni l'utente più idoneo.

Bibliografia

[1] Redazione di Rainews. Cani abbandonati, la piaga di tutte le estati: più di 3mila solo a luglio. Lo spot di Ornella Muti, August 2025.

[2] Animal Friends - Adopt a Dog.

[3] PetRescue. Pet rescue quiz.

[4] Cerca Pet: trova la razza di cane per te | Purina.

[5] Che cos'è un motore di raccomandazione? | IBM, June 2024.

[6] Charu C. Aggarwal. *Recommender systems: the textbook*. Springer, Cham Heidelberg New York Dordrecht London, 2016.

[7] Django.

[8] Introduzione al web framework Django e Primo Progetto.

[9] Django introduction - Learn web development | MDN, June 2025.

[10] Angelo Gentile III. Basics of Django: Model-View-Template (MVT) Architecture, March 2024.

[11] Shared Nothing Architecture: Pros, Cons & Best Practices.

[12] Come funziona l'architettura M.T.V. di Django?

[13] Tony Chan. *Django: Il framework web ad alto livello per Python*. Tony Chan, 2025. Printed in Poland by Amazon Fulfillment Poland Sp. z o.o., Wroclaw; 18 May 2025.

[14] Views In Django | Python, May 2019.

[15] Django shortcut functions | Django documentation.

[16] 05. Architettura MTV, Urls e Views - Programmare in Python.

[17] Flask vs. Django: framework Python per lo sviluppo di applicazioni web, December 2023.

[18] Django VS Ruby on Rails: quale framework scegliere?

[19] RanjitPal Singh. Ruby on Rails Vs Django: quale framework è il migliore per il 2023?, January 2023.

[20] Durga Prasad Acharya. Django vs Laravel: Qual è il Framework Migliore nel 2025?, October 2021.

[21] Cos'è Docker? | IBM, June 2024.

[22] Che cos'è Docker?

[23] Dockerfile: Tutti i comandi, June 2020.

[24] Volumes, July 2025.
[25] Docker vs. virtual machines, February 2025.
[26] Docker vs. Virtual Machines: Exploring Key Differences and Ideal Use Cases, May 2025.
[27] Serena Sensini. *Docker: sviluppare e rilasciare software tramite container*. Apogeo, Milano, 2020. OCLC: 1277310262.
[28] Che cos'è un database?
[29] Database - Introduzione.
[30] Relational Model.
[31] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, June 1970. In: *Information Retrieval*, P. Baxendale (ed.).
[32] Codd's 12 Rules.
[33] SQL vs NoSQL: Le differenze spiegate | OVHcloud Italia.
[34] Cloudflare. Sql injection | cos'è l'sql injection?, 2025. [Online; accesso 12 Ottobre 2025].
[35] Mark Otto, Jacob Thornton, and Bootstrap contributors. Get started with Bootstrap.
[36] Introduzione a Bootstrap.
[37] Cos'è una CDN (Content Delivery Network)? | IBM, October 2025.
[38] Package manager o gestori pacchetti Linux: cosa sono e quali le differenze.
[39] Responsive Web Design, la guida.
[40] What Is Cosine Similarity? | IBM, July 2025.
[41] Cos'è: Distanza euclidea.
[42] Alessandro Catini. Correlazione lineare di Pearson, May 2023.
[43] Il coefficiente di correlazione lineare di Pearson.
[44] Mayurdhvajsinh Jadeja. Jaccard Similarity Made Simple: A Beginner's Guide to Data Comparison, March 2024.
[45] similarities module — Surprise 1 documentation.
[46] Che cos'è un diagramma entità-relazione? | IBM, June 2024.

Ringraziamenti

Il più grande ringraziamento deve andare ai miei nonni, per avermi cresciuta e avermi fatto da genitori, punto di riferimento, e poi nonni. Grazie a mio nonno per essersi preso cura della me bambina più di chiunque altro. Grazie a mia nonna per avermi accompagnata e spronata ad andare avanti finché ne ha avuto la possibilità. Questo percorso non ha lo stesso significato senza di lei. Ma spero che da ovunque si trovino entrambi possano vedermi, non tanto per essere orgogliosi di me, di quello non ho mai dovuto dubitare anche quando non portavo il risultato sperato, ma per poter godere di un traguardo che abbiamo raggiunto insieme. Non sarei la stessa persona se non avessi avuto grazie a loro quel senso di famiglia che mio nonno con la sua sola presenza riusciva a garantire e che avrei voluto durasse in eterno. Non sarebbe stato lo stesso percorso se non avessi potuto contare sugli "in bocca al lupo" di mia nonna prima di ogni esame, l'unica persona che sapeva le date dei miei esami e anche la stessa persona che si agitava più di me per poi festeggiare i miei successi o tranquillizzarmi e spronarmi a riprovarci quando le cose non andavano nel modo migliore. È alle due persone che mi hanno educata, sgridata senza mai sentirli alzare la voce e che mi hanno dato un senso di sicurezza che va tutto il mio amore, rispetto ed il più grande grazie. Questo percorso è tutto per voi.

In seguito vorrei ringraziare me stessa, la persona complicata che sono, per essere sempre andata avanti, qualsiasi cosa capitasse o letteralmente mi crollasse addosso.

Un sincero grazie va ai miei amici, che si possono contare sulle dita di una mano ma che anche singolarmente hanno un valore inestimabile. Grazie Andrea ed Annachiara per esserci stati nel momento peggiore, e grazie per avermi pian piano riportato alla realta. Grazie Katia per esserci sempre stata da quando ti conosco, per essere la forte spalla su cui so di poter contare e per essere la persona che chiamo se rischio di scoppiare a piangere, perché so che entro metà chiamata mi starai già facendo ridere.

Grazie a Desy, a suo padre e a Marcello per il supporto e l'aiuto su cui mi hanno permesso di contare e per avermi trattata fin da subito come parte della famiglia.

Grazie a tutti coloro che hanno fatto il tifo per me e che mi hanno realmente sostenuta.

Grazie a mio padre per avermi sostenuto e finanziato gli studi e per essere stato capace di mettersi in discussione.

Grazie a mio fratello per le risate, le discussioni, le pazienze infinite e quelle finite.

Infine, grazie Desy. I motivi di questo grazie sono molti e difficili da spiegare: grazie per la pazienza, che sotto sotto in effetti hai; grazie per la comprensione; grazie per avermi spronato ad evolvere; grazie per i doppi in bocca al lupo; grazie per esserci stata, anche se non te l'ho reso per niente facile; grazie per aver compreso le mie emozioni prima ancora che io decidessi se e quando elaborarle; grazie per tutte le canzoni che odi e che mi mettevi spontaneamente quando iniziavo a chiudermi; e molto altro che non saprei come mettere nero su bianco, ma per cui un grazie sarebbe riduttivo, quindi mi limiterò a dire: telepaticamente... sempre.