SCHNAUZER GEO

Tesina del progetto del corso di tecnologie Web || A.A. 2024-2025

Giovanna Mantovani – MATRCOLA: 146541

Sommario

[1. Presentazione generale del progetto 3](#_Toc184819789)

[**1.1 Traccia** 3](#_Toc184819790)

[**1.2** **Funzionalità:** 3](#_Toc184819791)

[**1.3** **Use Case Diagrammi UML:** 5](#_Toc184819792)

[**1.4 Git** 5](#_Toc184819793)

[2. Database 6](#_Toc184819794)

[**2.1 Scelta del Database** 6](#_Toc184819795)

[**2.2 Schema delle Classi** 6](#_Toc184819796)

[**2.3.1 La classe Utente** 7](#_Toc184819797)

[**2.3.2 Prenotazioni** 8](#_Toc184819798)

[**2.3.3 Recensioni** 9](#_Toc184819799)

[3. Logica backend 9](#_Toc184819800)

[**3.1 I Packages** 10](#_Toc184819801)

[4. Logica Frontend 10](#_Toc184819802)

[**4.1 Navbar e Ricerca Globale** 10](#_Toc184819803)

[**4.1.1 View “search\_results”** 11](#_Toc184819804)

[**4.2 Gestione delle Prenotazioni** 11](#_Toc184819805)

[**4.3 Le mie Prenotazioni** 11](#_Toc184819806)

[**4.3.1 Struttura del template** 12](#_Toc184819807)

[**4.3.2 Interazione con il Backend** 12](#_Toc184819808)

[**4.4 Recensioni** 12](#_Toc184819809)

[**4.4.1 Struttura del template** 12](#_Toc184819810)

[**4.4.*2* Interazione con il Backend** 13](#_Toc184819811)

[5. Sfide Affrontate 13](#_Toc184819812)

[**5.1 Creazione del Form di Prenotazione** 13](#_Toc184819813)

[**5.2 Sfida: Filtraggio e Ordinamento delle Prenotazioni** 14](#_Toc184819814)

[**5.3 Sfida: Ricerca Globale dei cuccioli** 14](#_Toc184819815)

[Durante lo sviluppo della funzionalità di ricerca, ho incontrato un problema in cui il sistema non restituiva risultati al primo click, nonostante i dati fossero presenti nel database. Questo accadeva solo alla prima ricerca, mentre la seconda restituiva i risultati corretti. Il problema era causato da una gestione errata del parametro di ricerca e dalla mancanza di controllo per una query vuota o un input non valido. 14](#_Toc184819816)

[Per risolvere, ho modificato il codice Python per recuperare il parametro con il nome q dalla richiesta GET. Questa modifica ha permesso di allineare il campo di input con la variabile nel backend, garantendo che i risultati venissero filtrati correttamente. Inoltre, ha risolto il problema della query vuota, mostrando tutti i risultati quando l'utente non inserisce alcun termine di ricerca. 14](#_Toc184819817)

[6. Test 15](#_Toc184819818)

[7. Screenshots 15](#_Toc184819819)

## **Presentazione generale del progetto**

### **1.1 Traccia**

Schnauzer Geo è un sito web sviluppato con Django, pensato per gli appassionati di Schnauzer.

Offre una piattaforma interattiva che permette agli utenti di scoprire i cuccioli disponibili, suddivisi per taglia (nano, standard, gigante), e approfondire le caratteristiche della razza.

Previa registrazione, gli utenti possono prenotare corsi di addestramento per i loro cani, scegliendo comodamente data e orario in base alla disponibilità.

Grazie al sistema di gestione delle prenotazioni, gli utenti registrati possono controllare e annullare le proprie richieste in qualsiasi momento.

È inoltre possibile lasciare recensioni sui servizi ricevuti, contribuendo così a creare una community sulla condivisione di esperienze.

### **1.2 Funzionalità:**

* Esplorazione cuccioli: Gli utenti, registrati o ospiti, possono esplorare i cuccioli di Schnauzer presenti nella piattaforma, suddivisi in tre categorie principali:
  + **Nano**: Cuccioli di taglia piccola.
  + **Standard**: Cuccioli di taglia media.
  + **Gigante**: Cuccioli di taglia grande.

Per ciascuna categoria, gli utenti possono:

* + Ordinare i cuccioli in base a criteri personalizzati come prezzo e altri attributi.
  + Filtrare i cuccioli sulla base di parole chiave nella descrizione o nel nome utilizzando una barra di ricerca globale.
* Sistema di prenotazione: Gli utenti registrati possono prenotare lezioni di addestramento per il proprio cucciolo. La funzionalità include:
  + Selezione di corsi: Gli utenti possono scegliere il corso desiderato dalla sezione dedicata all'addestramento.
  + Prenotazione personalizzata: Per ogni corso, è possibile selezionare una data e un orario. Gli orari già prenotati da altri utenti sono bloccati per evitare conflitti.
  + Validazione dinamica: Il sistema controlla automaticamente che la data selezionata non sia già trascorsa o che l'orario sia disponibile, generando notifiche di errore in caso contrario.
  + Visualizzazione dinamica AJAX: Gli utenti possono visualizzare in tempo reale gli orari disponibili per una data specifica, grazie a richieste AJAX.

Una volta completata la prenotazione, l'utente riceve un messaggio di conferma e può consultare la propria lista di prenotazioni nella sezione personale.

* Gestione Prenotazioni: Gli utenti possono gestire le prenotazioni nella sezione "Mie Prenotazioni". Le opzioni disponibili includono:
  + Visualizzazione personalizzata: È possibile ordinare le prenotazioni per data o prezzo, sia in ordine crescente che decrescente.
  + Filtraggio attivo: Gli utenti possono scegliere di visualizzare solo le prenotazioni attive.
  + Cancellazione: È possibile eliminare prenotazioni già effettuate o annullare quelle non più desiderate. Una notifica è generata per confermare l'annullamento.
* Sistema di Autenticazione: La piattaforma implementa un sistema completo di gestione degli utenti basato sulle funzioni standard di Django:
  + Registrazione: Gli utenti possono creare un nuovo account attraverso un modulo personalizzato.
  + Login e Logout: Autenticazione semplice tramite username e password.
  + Controllo accesso: Alcune sezioni, come la prenotazione o la gestione degli account, sono accessibili solo agli utenti autenticati.
* Recensioni: Gli utenti autenticati possono lasciare recensioni sui servizi utilizzati. Le recensioni possono includere:
  + Voto numerico.
  + Testo descrittivo.
  + Una foto opzionale.

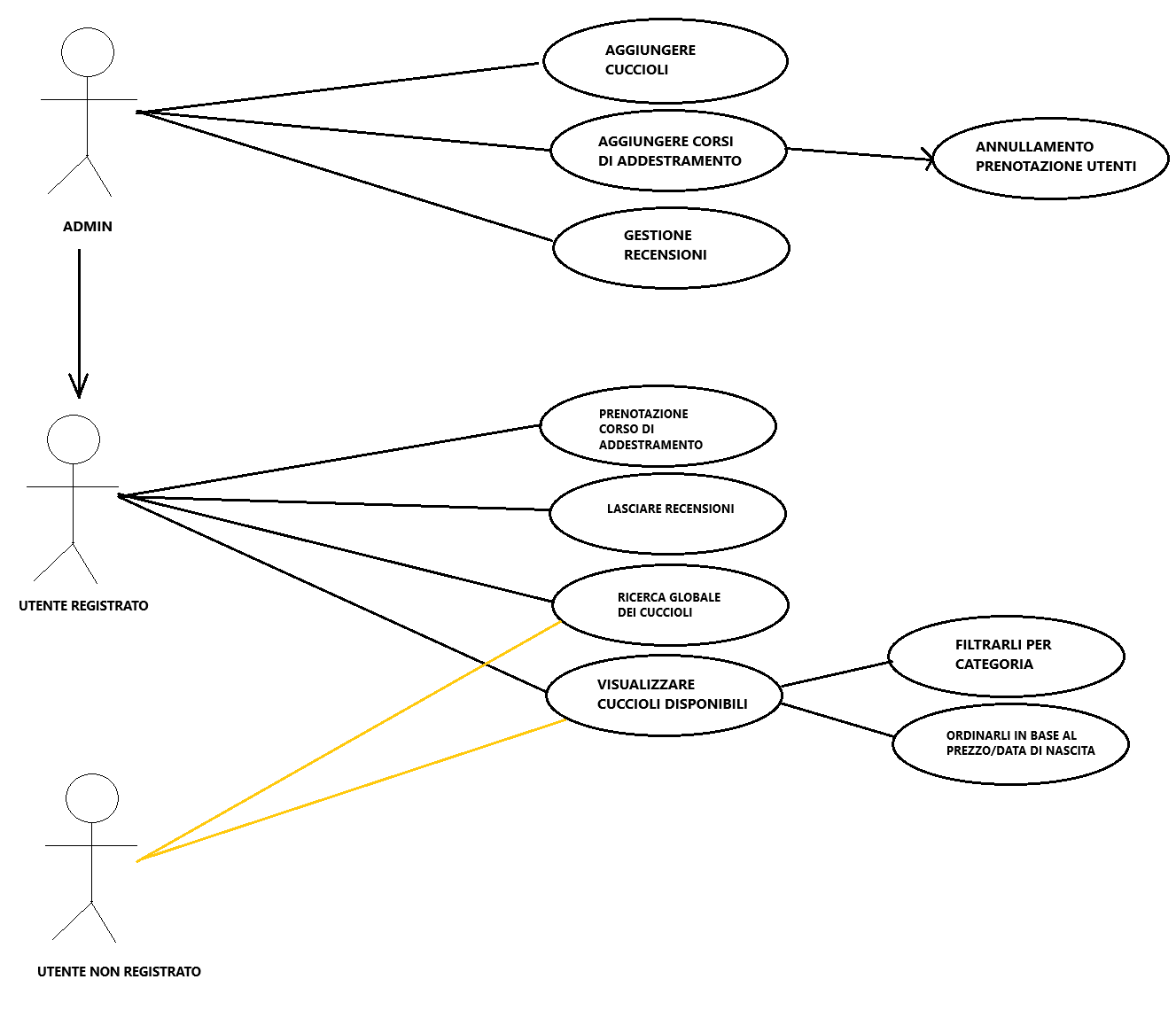
Tutte le recensioni sono salvate e visualizzate in ordine cronologico decrescente, favorendo la visibilità di quelle più recenti.

* Funzionalità di Ricerca: Gli utenti possono utilizzare una barra di ricerca globale per trovare cuccioli o corsi di addestramento specifici. Il sistema supporta:
  + Ricerca basata su nome o descrizione.
  + Query parziali o complete.

I risultati sono filtrati dinamicamente e presentati in una pagina dedicata.

### **1.3 Use Case Diagrammi UML:**

Di seguito il diagramma:



### **1.4 Git**

Per l’hosting remoto ho scelto di utilizzare la piattaforma Github. Il repositery si trova all’indirizzo: <https://github>.com/mantovanigiovi/Allevamento

## **2. Database**

### **2.1 Scelta del Database**

Per questo progetto è stato utilizzato il database SQLite, predefinito in Django e ideale per lo sviluppo e il collaudo delle applicazioni. Nonostante questa scelta sia pratica durante la fase iniziale, per ambienti di produzione si raccomanda l'adozione di un database più robusto, come MySQL o PostgreSQL, per garantire migliori prestazioni e affidabilità. Per implementare questa modifica, è necessario configurare il file settings.py.

### **2.2 Schema delle Classi**

Di seguito è riportato il diagramma delle classi generato tramite **graph\_models**, che illustra la struttura delle entità del database e le relazioni tra di esse. Questo schema è stato progettato per rappresentare i principali modelli definiti nell’applicazione, evidenziando le connessioni tra tabelle e attributi principali.

Immagine che contiene testo, diagramma, Piano, schematico

Descrizione generata automaticamente

### **2.3.1 La classe Utente**

La prima macro-entità su cui si basa il progetto è l'utente registrato al sistema. La gestione degli utenti è affidata al modello predefinito User di Django, che consente di autenticare, registrare e gestire gli utenti all'interno dell'applicazione.

In questo progetto, l'utente ha un ruolo centrale: può registrarsi, effettuare login e accedere a diverse funzionalità, come la prenotazione di corsi di addestramento per Schnauzer o la scrittura di recensioni. Inoltre, l'utente può visualizzare e gestire le proprie prenotazioni. Alcuni campi personalizzati, come il nome, cognome e taglia del cane, vengono richiesti durante le prenotazioni per offrire un'esperienza più specifica e "personalizzata"

Per migliorare la gestione delle funzionalità disponibili, il progetto sfrutta il sistema di decoratori di Django. Ad esempio, l'accesso alla pagina **"mie prenotazioni"** è limitato esclusivamente agli utenti autenticati grazie al decoratore @login\_required. Questo garantisce che i dati delle prenotazioni siano accessibili solo all'utente proprietario.

Immagine che contiene testo, schermata, biglietto da visita

Descrizione generata automaticamente

In questo caso non è possibile la visualizzazione della pagina ‘Recensioni’, in quanto è esclusiva agli utenti registrati.

Ogni utente registrato può vedere notifiche personalizzate relative a modifiche o annullamenti di prenotazioni. Questo è reso possibile dal modello Notifica, che memorizza messaggi univoci per ciascun utente.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

### **2.3.2 Prenotazioni**

Una delle funzionalità principali del progetto è la gestione delle prenotazioni per i corsi di addestramento. La classe Prenotazione è progettata per raccogliere tutte le informazioni necessarie: nome, cognome, taglia del cane, data e orario della lezione, corso scelto, e stato della prenotazione (attiva o annullata).

Immagine che contiene testo, software, schermata

Descrizione generata automaticamente

Per garantire un'interfaccia intuitiva, il sistema permette agli utenti di selezionare solo orari disponibili in base a una data scelta. Grazie ad una logica implementata nella vista prenota, vengono filtrati gli orari già occupati:

### **2.3.3 Recensioni**

Per incentivare la partecipazione attiva degli utenti, è stata implementata una sezione dedicata alle recensioni. Ogni utente autenticato può lasciare un feedback sui servizi ricevuti, aggiungendo eventualmente un'immagine per documentare la propria esperienza.

La classe Recensione memorizza:

Immagine che contiene schermata, testo, software, Pagina Web

Descrizione generata automaticamente

* Il voto da 1 a 5.
* Un testo descrittivo.
* Una foto facoltativa.
* La data di creazione.

## **3. Logica backend**

Le cartelle del progetto sono organizzate come segue. Quelle principali vengono dettagliate di fianco:

Immagine che contiene testo, schermata, Carattere, diagramma

Descrizione generata automaticamente

* Allevamento è la cartella creata di default da startproject;
* docs contiene la tesina, la traccia e gli screenshot del progetto;
* Geo contiene il codice principale.
* migration contiene le migrazioni generate durante lo sviluppo del sito;
* static contiene le immagini usate per la sezione “razza” e il logo.
* templates contiene i templates HTML;
* media contiene le immagini dei cuccioli disponibili e le immagini dei corsi di addestramento.

### **3.1 I Packages**

Nel mio progetto utilizzo un approccio minimale alle dipendenze esterne. Come mostrato nel file requirements.txt, la sola libreria richiesta è **Django**.

Ciò riduce la complessità del progetto, rendendolo più facile da configurare e minimizzando i rischi di conflitti tra librerie o altri problemi di compatibilità.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

## **4. Logica Frontend**

Per il frontend del progetto ho utilizzato un approccio basato su HTML, CSS e il motore di template di Django. Ho fatto uso di Bootstrap per il design responsivo e di alcune personalizzazioni di stile per migliorare l'esperienza utente. La struttura principale dei template è stata suddivisa in sezioni riutilizzabili per garantire modularità e semplicità nella gestione del codice.

### **4.1 Navbar e Ricerca Globale**

Per migliorare l’usabilità e permettere agli utenti di navigare e cercare in modo efficiente, ho implementato una barra di navigazione che include una funzionalità di ricerca globale.

* **Template Coinvolti:** La navbar è inclusa in tutti i template principali tramite {% include 'Geo/navbar.html' %}. La casella di ricerca è integrata nella navbar, permettendo agli utenti di cercare direttamente i contenuti desiderati, come cuccioli o corsi.
* **Risultati della ricerca:** I risultati vengono visualizzati nel template search\_results.html, dove ogni risultato viene presentato come una card responsiva.

### **4.1.1 View “search\_results”**

La ricerca globale è gestita dalla view search\_results, che accetta richieste GET. La query string viene utilizzata per filtrare il database dei cuccioli o dei corsi. I risultati sono case-insensitive e vengono visualizzati con il seguente layout:

* Immagini ottimizzate con altezza limitata a 200px.
* Informazioni come nome, descrizione e prezzo visualizzate in formato card.
* Collegamenti diretti ai dettagli dei risultati tramite il pulsante “Dettagli”.

Immagine che contiene testo, schermata, Sito Web, Pagina Web

Descrizione generata automaticamente

### **4.2 Gestione delle Prenotazioni**

La gestione delle prenotazioni consente agli utenti di visualizzare, ordinare e filtrare le proprie prenotazioni attraverso un'interfaccia interattiva.

* **Template Coinvolti:** La logica è suddivisa tra Geo/navbar.html (per la navigazione) e un template principale specifico, mie\_prenotazioni.html, che presenta le prenotazioni in un formato a card.
* **Funzionalità Principali:**
  + Ordinamento delle prenotazioni in base a data o prezzo.
  + Visualizzazione condizionata per prenotazioni attive o annullate.
  + Messaggi di notifica dinamici con il sistema di messaggi di Django.
  + Pulsanti di azione per eliminare o rimuovere notifiche.

### **4.3 Le mie Prenotazioni**

Il template “le\_mie\_prenotazioni” è pensato per fornire agli utenti una panoramica delle loro prenotazioni, con la possibilità di ordinarle o filtrare quelle attive.

### **4.3.1 Struttura del template**

**Introduzione e Messaggi Personalizzati**:

* Saluto personalizzato basato sull'utente autenticato, mostrato tramite

{{ user.username }}.

* Messaggi di sistema come notifiche di successo o avvisi di errori, gestiti con

 {% if messages %}.

**Filtri di Ordinamento**:

* Il modulo in cima alla pagina consente agli utenti di ordinare le prenotazioni per:
  + Data (più vicina o più lontana).
  + Prezzo (crescente o decrescente).
* È possibile filtrare solo le prenotazioni attive tramite un checkbox.

**Lista delle Prenotazioni**:

* Ogni prenotazione viene mostrata in una **card** con:
  + Corso e altre informazioni principali (es. data, orario, taglia).
  + Stato della prenotazione (es. annullata).
  + Pulsante di eliminazione.

### **4.3.2 Interazione con il Backend**

Il template utilizza cicli {% for prenotazione in prenotazioni %} per mostrare dinamicamente le prenotazioni. Ogni card include anche una gestione per:

* **Prenotazioni Annullate**:
  + Se una prenotazione è annullata ({% if prenotazione.is\_annullata %}), viene mostrato un messaggio specifico.
  + L'utente può eliminare una notifica relativa a una prenotazione annullata tramite un pulsante che invia una richiesta POST.
* **Pulsante di Eliminazione:**
  + Ogni card contiene un link con un'azione collegata a una view Django ({% url 'elimina\_prenotazione' prenotazione.id %}), previa conferma tramite un popup JavaScript.

### **4.4 Recensioni**

Il template recensioni è stato progettato per gestire le recensioni degli utenti. Include funzionalità per aggiungere una nuova recensione e visualizzare quelle esistenti.

### **4.4.1 Struttura del template**

**Modulo per Inserire una Recensione**:

* Il form consente agli utenti di:
  + Lasciare una valutazione da 1 a 5 stelle utilizzando un sistema di selezione visivo.
  + Inserire un commento tramite un campo textarea.
  + Caricare una foto opzionale per arricchire la recensione.
* **Gestione delle Stelle**:
  + Il sistema di valutazione utilizza una classe .star-rating con un design intuitivo che cambia il colore delle stelle in base al valore selezionato, grazie all'uso di CSS e pseudo-classi (hover, checked).

**Visualizzazione delle Recensioni Precedenti**:

* Ogni recensione è mostrata con dettagli come:
  + Utente che ha lasciato il commento.
  + Valutazione rappresentata graficamente con stelle.
  + Commento testuale.
  + Foto (se presente), mostrata tramite l'elemento <img>.

### **4.4.*2* Interazione con il Backend**

La gestione del form avviene tramite il metodo POST, e il template si avvale del token {% csrf\_token %} per garantire la sicurezza delle richieste. Le recensioni vengono iterate attraverso un ciclo {% for recensione in recensioni %}, e ogni dato viene recuperato direttamente dal modello passato dal backend.

## **5. Sfide Affrontate**

Durante lo sviluppo del sito web mi sono imbattuta in diverse sfide tecniche e progettuali specialmente per quanto riguarda la gestione delle prenotazioni di addestramento per cani e la barra di ricerca. Nonostante il progetto si sia rivelato stimolante e mi abbia permesso di apprendere molto, alcune problematiche sono emerse in modo particolare nel corso del lavoro. Trattandosi della mia prima esperienza con Django, ho dovuto imparare l’utilizzo di modelli, viste e template, oltre che a gestire interazioni più avanzate con l'utente. Di seguito descrivo le difficoltà principali e come le ho affrontate.

### **5.1 Creazione del Form di Prenotazione**

Implementare un form di prenotazione funzionale per i corsi di addestramento è stato più complesso del previsto. Inizialmente, il problema principale era definire una struttura che permettesse agli utenti di selezionare i dettagli necessari, come il corso, la data, l’orario e altre informazioni, garantendo al contempo che i dati inseriti fossero validi e coerenti.

**Problemi riscontrati:**

1. Gestione della disponibilità degli orari  
   La prima grande difficoltà è stata gestire la disponibilità degli orari in base alla data scelta dall’utente. La logica richiesta era di controllare dinamicamente quali orari fossero già occupati per ogni corso specifico e nascondere le opzioni non valide nel form. Questo ha richiesto l’utilizzo di richieste AJAX per aggiornare il menu degli orari senza dover ricaricare la pagina.
2. Validazione dei dati  
   Un’altra sfida è stata garantire che i dati immessi dagli utenti fossero corretti e coerenti. Ad esempio, verificare che una data passata non potesse essere selezionata o che il formato dell’email fosse valido. Questo è stato gestito tramite una combinazione di validazioni lato server (nei metodi clean del form) e lato client (con JavaScript).

3.Interfaccia Utente  
Creare un’interfaccia user-friendly che fosse al tempo stesso chiara e completa è stato più complicato del previsto. Ho cercato di bilanciare l’estetica con la funzionalità, utilizzando uno stile semplice ma efficace, con pulsanti, menù a discesa e messaggi di feedback ben visibili.

**Soluzioni Implementate:**

* Richieste AJAX per la selezione degli orari  
  Ho utilizzato JQuery per inviare richieste al server ogni volta che l’utente selezionava una data. Questo mi ha permesso di aggiornare il menu a discesa degli orari disponibili in tempo reale, utilizzando viste Django che restituiscono dati in formato JSON.
* Separazione della logica di calcolo  
  Per evitare di sovraccaricare il form stesso, ho separato la logica di controllo degli orari e dei dati in diverse viste e funzioni. Ad esempio, una vista si occupa esclusivamente di calcolare gli orari disponibili (calcola\_orari), mentre un’altra valida i dati della prenotazione.
* Utilizzo di modelli relazionali  
  Il modello Prenotazione è stato progettato in modo da collegarsi a più entità, come l’utente e il corso, garantendo così una gestione centralizzata e coerente dei dati.
* Feedback migliorato  
  I messaggi sono stati integrati direttamente nei template e resi visibili con stili chiari. Ad esempio, un errore nella selezione di un orario viene mostrato immediatamente con un messaggio evidenziato in rosso.

### **Sfida: Filtraggio e Ordinamento delle Prenotazioni**

1. Filtraggio delle Prenotazioni Attive  
   La necessità di mostrare solo le prenotazioni "attive" ha richiesto una logica specifica per distinguere queste ultime dalle prenotazioni "annullate" o concluse. Questo problema è diventato più complesso nel momento in cui era necessario mantenere questa funzionalità sincronizzata con l'interfaccia (checkbox).
2. Ordinamento Dinamico  
   Implementare un sistema di ordinamento in base a parametri diversi (data crescente, decrescente, prezzo crescente, ecc.) ha richiesto una gestione accurata dei parametri inviati dal frontend, evitando conflitti tra i vari criteri di ordinamento.
3. Gestione delle Query  
   Filtrare e ordinare efficacemente un numero elevato di prenotazioni senza compromettere le performance ha rappresentato una sfida, soprattutto nel bilanciare le query al database.

**5.3 Sfida: Ricerca Globale dei cuccioli**

Durante lo sviluppo della funzionalità di ricerca, ho incontrato un problema in cui il sistema non restituiva risultati al primo click, nonostante i dati fossero presenti nel database. Questo accadeva solo alla prima ricerca, mentre la seconda restituiva i risultati corretti. Il problema era causato da una gestione errata del parametro di ricerca e dalla mancanza di controllo per una query vuota o un input non valido.

Per risolvere, ho modificato il codice Python per recuperare il parametro con il nome q dalla richiesta GET. Questa modifica ha permesso di allineare il campo di input con la variabile nel backend, garantendo che i risultati venissero filtrati correttamente. Inoltre, ha risolto il problema della query vuota, mostrando tutti i risultati quando l'utente non inserisce alcun termine di ricerca.

## **6. Test**

Ho effettuato test su:

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

I test sono visualizzabili al file Allevamento/Geo/test.py

## **7. Screenshots**

Gli screenshot, compresi quelli utilizzati in questa tesina, sono disponibili in docs/screenshot