Panoramica progetto **1**
Funzionalità generali 1
Funzionalità per utenti registrati 2
Funzionalità per proprietari di ristoranti 2
Gestione account 2
USE CASE UML: Funzionalità per tipologia utente **3**
MODELLI UTILIZZATI **6**
DIAGRAMMA ER **9**
Relazioni tra le due app 13
ADMIN **13**
POPOLAZIONE DATABASE **14**
FUNZIONE NOTIFICA **15**
ESEMPIO GRAFICI PER PROPRIETARIO **16**
TEST EFFETTUATI **17**
PROBLEMI BUG RISCONTRATI **18**

Panoramica progetto

Il progetto consiste in un sito web realizzato in Django, progettato per offrire una piattaforma user-friendly dedicata alla gestione di ristoranti e prenotazioni. Le funzionalità principali possono essere suddivise in base ai ruoli degli utenti (utenti registrati e proprietari di ristoranti) e alle interazioni disponibili tramite l'interfaccia grafica.

Funzionalità generali

- **Ricerca e filtraggio ristoranti**:
  Dalla homepage, un utente qualsiasi può ricercare ristoranti basandosi su una città specifica. È inoltre possibile filtrare i risultati per ottenere una lista mirata in base alla tipologia di ristorante desiderata. Ogni ristorante dispone di una scheda dettagliata con informazioni quali posizione geografica e recensioni degli utenti.

Funzionalità per utenti registrati

- **Registrazione e interazioni**:
  Gli utenti possono registrarsi come utenti normali o come proprietari di ristoranti.
- **Prenotazioni**:
  Gli utenti normali possono effettuare prenotazioni, disdire prenotazioni esistenti o mettersi in lista di attesa nel caso in cui il ristorante non abbia posti disponibili. Se si libera un posto (es. a seguito di una cancellazione), tutti gli utenti in lista di attesa per quella fascia oraria ricevono una notifica.
- **Recensioni**:
  È possibile lasciare recensioni sui ristoranti visitati, fornendo feedback utile per altri utenti.

Funzionalità per proprietari di ristoranti

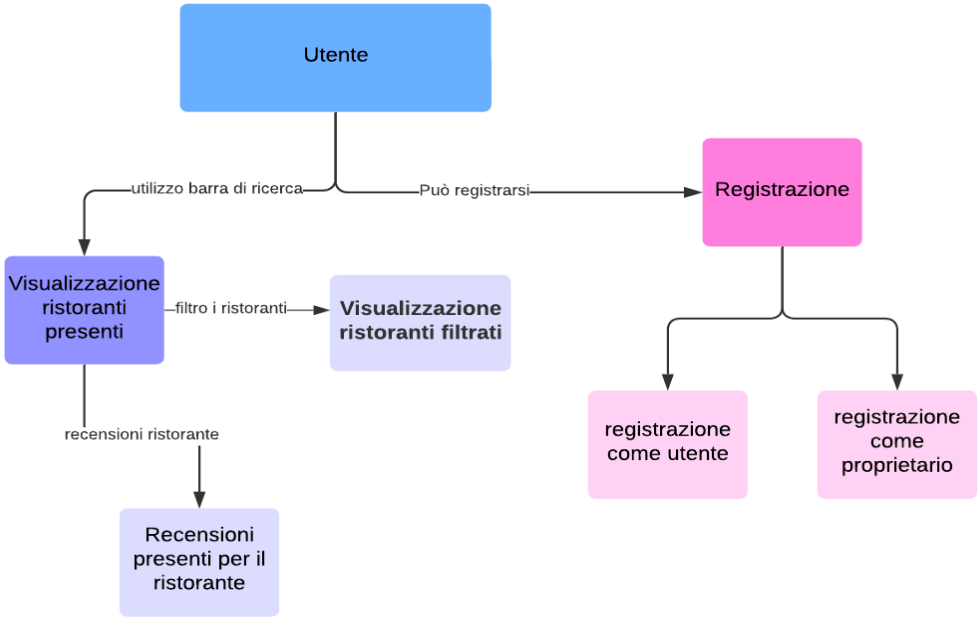
- **Gestione dei ristoranti**:
  Gli utenti registrati come proprietari possono aggiungere uno o più ristoranti, visualizzare le recensioni ricevute, e accedere a grafici analitici che mostrano l'andamento del loro business (es. trend di prenotazioni).
- **Notifiche sulle prenotazioni**:
  I proprietari ricevono notifiche ogni volta che viene effettuata una prenotazione per uno dei loro ristoranti.

Gestione account

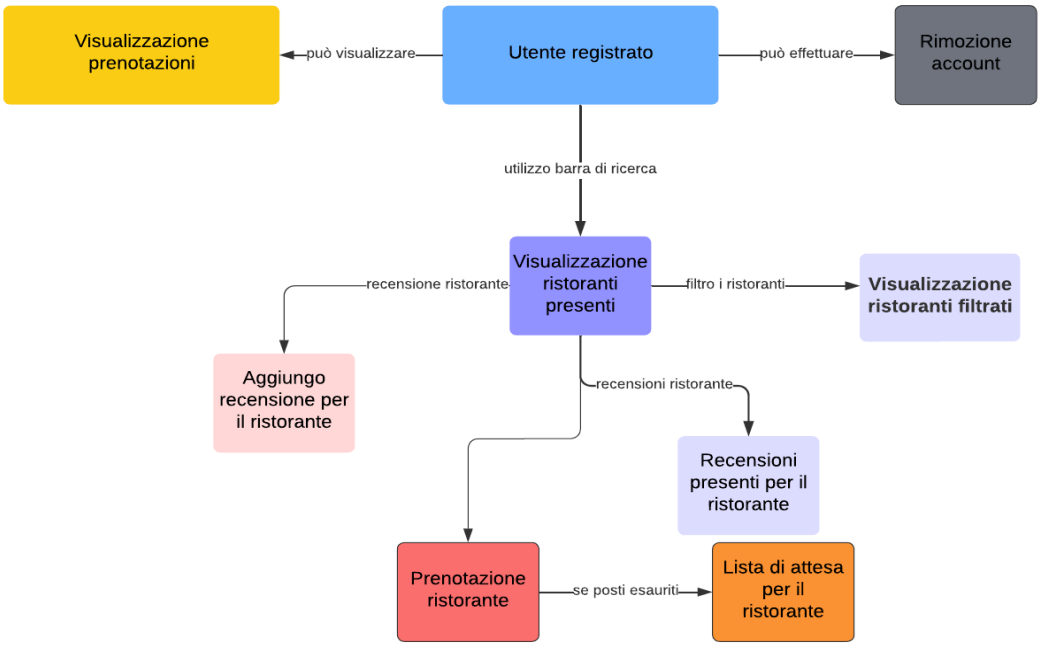
- Entrambi i tipi di utenti (normali e proprietari) possono decidere in qualsiasi momento di rimuovere il proprio account dal sistema.

USE CASE UML: Funzionalità per tipologia utente

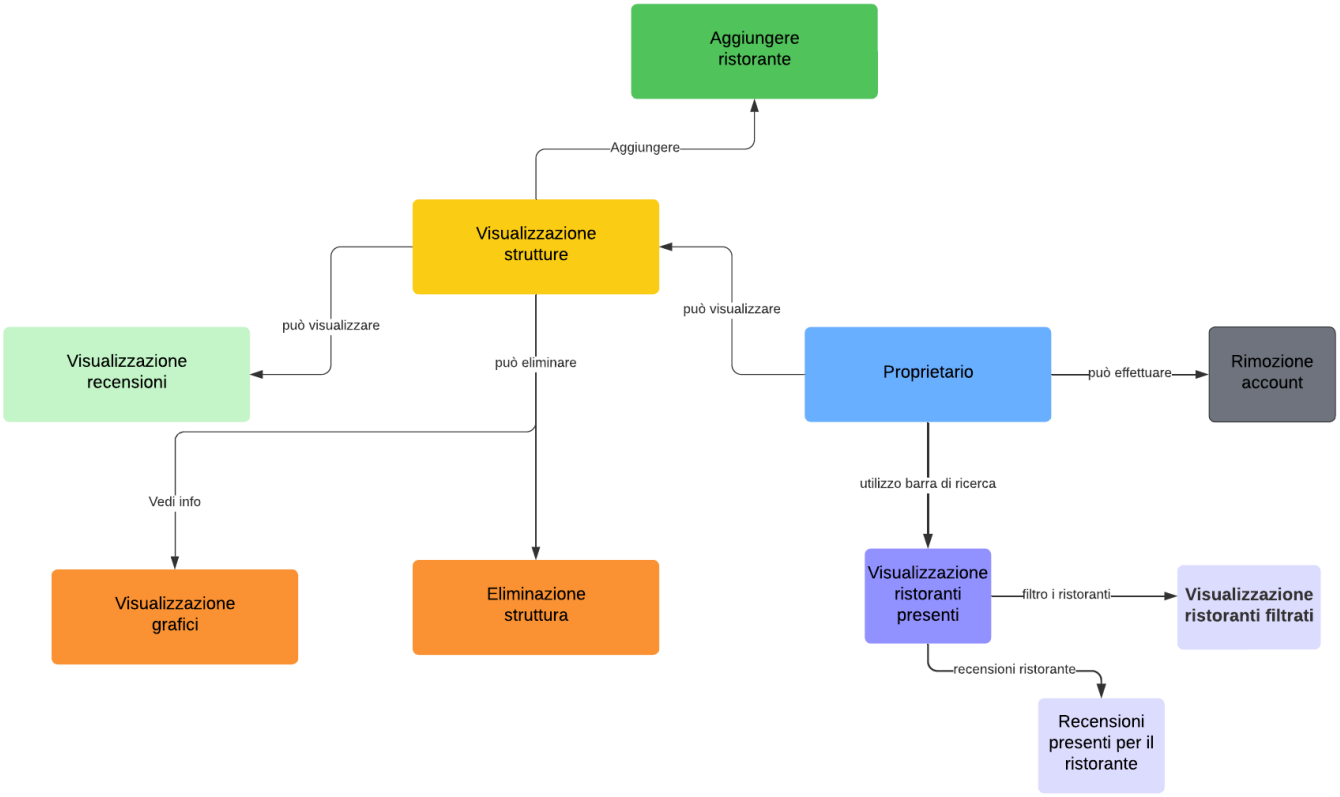
Utente non registrato



Utente registrato



Proprietario


Aggiungere ristorante
Aggiungere
Visualizzazione strutture
può visualizzare
Visualizzazione recensioni
può eliminare
può visualizzare
Proprietario
può effettuare
Rimozione account
Vedi info
utilizzo barra di ricerca
Visualizzazione grafici
Eliminazione struttura
Visualizzazione ristoranti presenti
filtro i ristoranti
Visualizzazione ristoranti filtrati
recensioni ristorante
Recensioni presenti per il ristorante

MODELLI UTILIZZATI

Il database scelto per la realizzazione del progetto è sqlite, già integrato con Django. È stato scelto per la sua facilità di utilizzo e semplicità d'uso, inoltre ha un portabilità notevole in quanto è sufficiente spostare il database da un ambiente ad un altro semplicemente copiando il file relativo al database sql: .sqlite3

I modelli sono stati progettati per riflettere la struttura logica dell'applicazione, con relazioni chiave tra entità come utenti, ristoranti, prenotazioni e recensioni. Ogni modello rappresenta una tabella nel database, seguendo il paradigma ORM (Object-Relational Mapping) di Django, che facilita l'interazione con il database tramite codice Python invece che query SQL dirette.

in seguito la descrizione di ciascun modello utilizzato:

1) CustomUser(AbstractUser):Il modello CustomUser estende AbstractUser di Django per permettere una personalizzazione dell'autenticazione utente. L'inclusione dei flag is_customer e is_owner consente di distinguere facilmente tra un utente normale e un proprietario di ristorante, semplificando la gestione dei ruoli e delle funzionalità specifiche.

2) Customer(models.Model): questo è il modello utilizzato per specificare un utente non proprietario, a parte una short_bio che viene aggiunta come parametro extra, vi è una relazione uno-a-uno con il modello CustomUser. Questa scelta di suddividere il customer è stata fatta per dividere concettualmente il customer da owner (che invece è semplicemente un AbstractUser)

3) Restaurant(models.Model): È il modello che identifica un ristorante, composto da i seguenti attributi:

```
total_seats = models.IntegerField(default=10)
owner = models.ForeignKey(CustomUser, on_delete=models.CASCADE)
city = models.CharField(max_length=255)
restaurant_name = models.CharField(max_length=255)
image = models.ImageField(upload_to='owners_photos')
```

```
start_lunch = models.TimeField()
address = models.CharField(max_length=255, null=True)
waiting_list = models.ManyToManyField(CustomUser, related_name="waiting_restaurants", blank=True)
end_lunch = models.TimeField()
start_dinner = models.TimeField(null=True)
end_dinner = models.TimeField(null=True)
tags = models.ManyToManyField(Tag, related_name='restaurants', blank=True)
```

total_seats: rappresenta il numero totale di posti disponibili per il ristorante, la logica implementata suddivide i posti disponibili tra pranzo e cena. per esempio un total_seats=40 significa che il ristorante ha 40 coperti disponibili sia per il pranzo che per la cena.
Owner: è la relazione che identifica il CustomUser, in questo caso il proprietario
city: la città dove si trova il ristorante
restaurant_name: il nome del ristorante
image: l'immagine del ristorante
start_lunch,end_lunch,statr_dinner,end_dinner: rappresentano gli orari di inizio e fine rispettivamente di pranzo e cena
waiting_list: è una relazione molti-a-molti con il CustomUser, questo perchè un utente può essere in lista di atesa per più ristoranti e un ristorante può avere liste di attesa per più utenti.
address: è l'indirizzo del ristorante, che viene utilizzato sfruttando un widget di google per mostrare la posizione del ristorante nella mappa di maps.
tags: relazione molti-a-molti con il modello Tag che rappresenta i filtri che si possono applicare ad un ristorante, la logica è la seguente: un ristorante può avere più tag(filtri), un filtro puo appartenere a diversi ristoranti.

4) Tag: è il modello che rappresenta i filtri dei ristoranti, composto semplicemente da una stringa name

```
class Tag(models.Model):
  name = models.CharField(max_length=50, unique=True)
```

5) review:è il modello che rappresenta una recensione inerente ad un ristorante

```
class Review(models.Model):
  rev_customer = models.ForeignKey(Customer, on_delete=models.CASCADE)
  review_value = models.IntegerField()
  review_text = models.TextField()
```

```
  review_res = models.ForeignKey(Restaurant, on_delete=models.CASCADE)
```

rev_customer: l'utente che ha scritto la recensione
rev_value: il valore della recensione (da 1 a 5)
review_text: il testo della recensione
review_res: il ristorante di riferimento della recensione

6) Reservation: è il modello che identifica una prenotazione effettuata

```
class Reservation(models.Model):
  customer = models.ForeignKey(Customer, on_delete=models.CASCADE)
  seats = models.IntegerField()
  res_datetime = models.DateTimeField(default=None)
  restaurant = models.ForeignKey(Restaurant, on_delete=models.CASCADE)
```

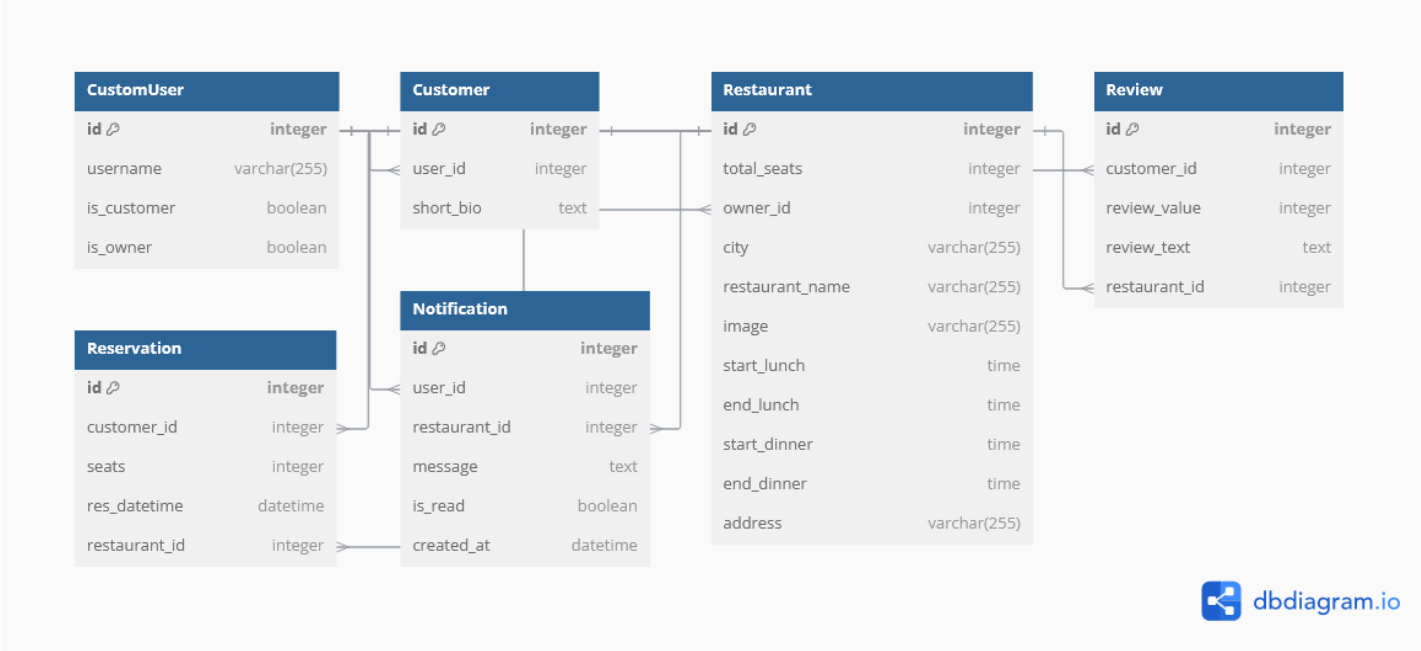
customer: l'utente che ha effettuato una prenotazione
seats: il numero di posti per la prenotazione
res_datetime: la data della prenotazione, in formato DateTime
restaurant: il ristorante di riferimento per la prenotazione

7) Notification: modello che rappresenta una notifica, che deve essere inviata in relazione ad una aggiunta in lista di attesa da parte di un utente o in relazione ad una prenotazione ad un proprietario

```
class Notification(models.Model):
  CustomUser = models.ForeignKey('CustomUser', on_delete=models.CASCADE,
related_name="notifications", null=True)
  restaurant = models.ForeignKey('Restaurant', on_delete=models.CASCADE,
related_name="notifications", null=True, blank=True)
  message = models.TextField()
  is_read = models.BooleanField(default=False)
  created_at = models.DateTimeField(auto_now_add=True)
```

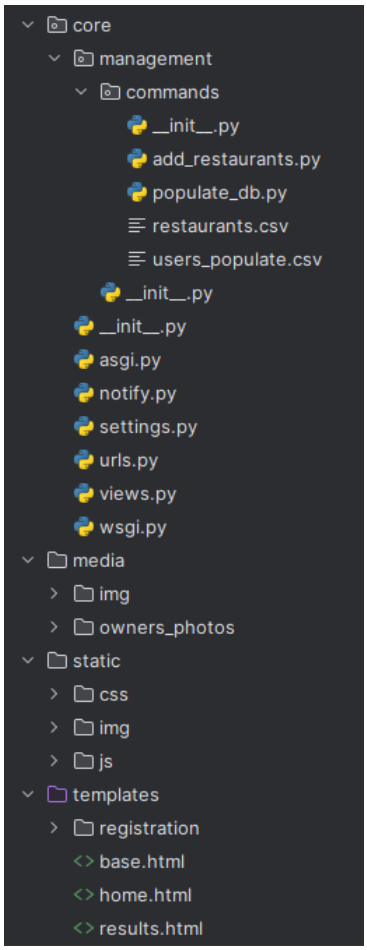
CustomUser: l'utente generico a cui è associata la notifica
restaurant: il ristorante di riferimento
message: il testo del messaggio da inviare
is_read: flag di verifica di lettura della notifica
created_at: data di creazione della notifica

DIAGRAMMA ER



Organizzazione logica applicazione:
Per la realizzazione del progetto sono state create 2 app con le seguenti funzioni logiche:

CORE

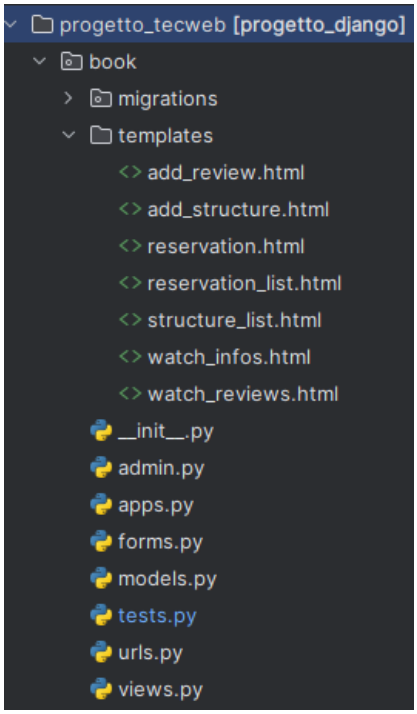


L’applicazione **CORE** rappresenta il nucleo del progetto e gestisce funzionalità comuni e la struttura fondamentale dell’applicazione.

- **Cartelle:**
    - **media**: Contiene tutte le immagini utilizzate nel progetto, come quelle dei ristoranti e degli utenti.
    - **static**: Contiene i file CSS, JavaScript e altre risorse statiche utilizzate per il front-end del sito. Tra i file più importanti ci sono il CSS per il design del sito e i file JavaScript per le funzionalità interattive.
    - **templates**: Contiene i file HTML che definiscono la struttura delle pagine. In particolare, troviamo:
        - **base.html**: Il template base che viene esteso da altre pagine.
        - **home.html e results.html**: Template relativi alla homepage e alla visualizzazione dei risultati di ricerca.
- **Cartelle e Funzionalità:**
    - **management/commands**: Contiene i comandi personalizzati che vengono utilizzati per la popolazione automatica del database tramite file CSV. Sono presenti file CSV per importare dati di utenti e ristoranti.
    - **registration**: Contiene i template per la gestione di login e registrazione degli utenti (moduli per il login e la creazione di un nuovo account).
    - **notify.py**: Gestisce l'invio di notifiche agli utenti, come quelle relative alle prenotazioni o alla lista di attesa per i ristoranti. Utilizza il modello Notification

L'app **CORE** si occupa quindi della gestione delle risorse statiche, dei modelli di base, della registrazione degli utenti e delle notifiche.

BOOK



L'applicazione **BOOK** è dedicata alla gestione delle funzionalità relative alle prenotazioni dei ristoranti, alle recensioni e alle liste d'attesa.

- **Funzionalità principali:**
  - **Prenotazioni**: Gestisce le prenotazioni degli utenti per i ristoranti, permettendo di visualizzare e modificare le prenotazioni tramite interfaccia utente.
  - **Recensioni**: Gli utenti possono scrivere recensioni sui ristoranti in cui hanno mangiato. Ogni recensione è associata a un ristorante specifico e a un utente.
  - **Lista d'attesa**: Gestisce la logica per la creazione e visualizzazione delle liste d'attesa per i ristoranti. Gli utenti possono essere aggiunti alle liste d'attesa per uno o più ristoranti.
  - **Ristoranti**: Gestisce tutte le informazioni relative ai ristoranti, come nome, posizione, orari di apertura e chiusura, e altre caratteristiche come i tag (filtri) associati ai ristoranti.

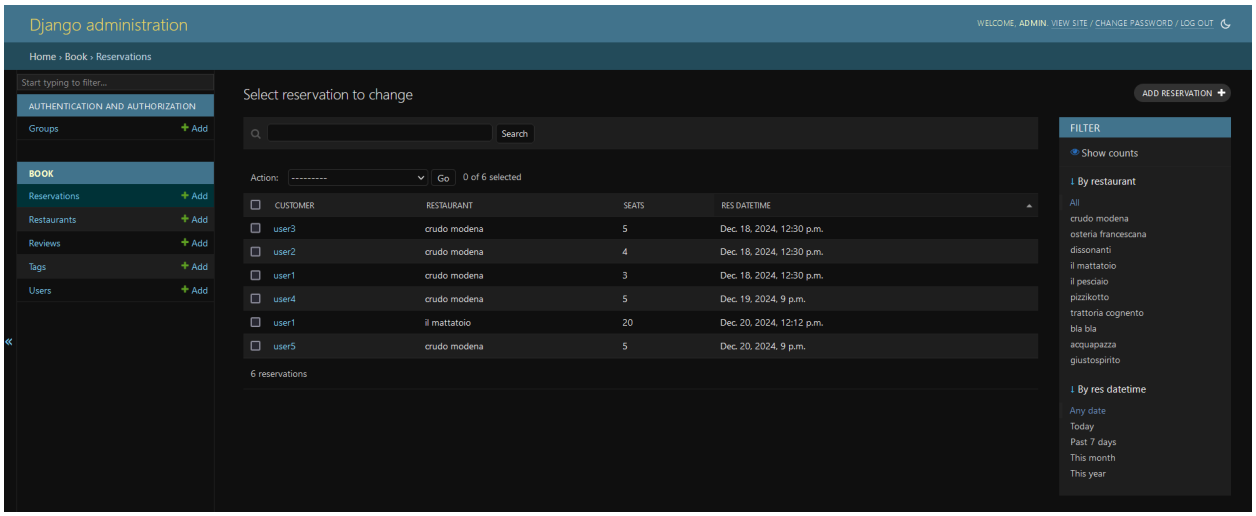
L'app **BOOK** interagisce con i modelli principali (come **Reservation**, **Review**, e **Restaurant**) per implementare queste funzionalità e garantire che tutte le operazioni siano registrate nel database in modo efficiente.

Relazioni tra le due app

- L'app **CORE** è il cuore dell'infrastruttura e delle funzionalità comuni, mentre **BOOK** si occupa della logica specifica per la gestione delle funzionalità

ADMIN

La sezione di admin, permette un controllo completo da parte dell'admin sulle funzionalità principali, come gestione delle prenotazioni, recensioni, ristoranti, tag, e utenti.



In questo esempio viene mostrata l'interfaccia di admin inerente alle prenotazioni, un admin può infatti cancellare o aggiungere una prenotazione da parte di un determinato utente.
Questo è il codice realizzato per la sezione di admin relativa alle prenotazioni

```python
@admin.register(Reservation)
class ReservationAdmin(admin.ModelAdmin):
    list_display = ('customer', 'restaurant', 'seats', 'res_datetime')
    list_filter = ('restaurant', 'res_datetime')
    search_fields = ('customer__user__username', 'restaurant__restaurant_name')
    ordering = ('res_datetime',)

    matteo pellacani
    def delete_model(self, request, obj):
        waiting_list_customers = obj.restaurant.waiting_list.all()
        for customer in waiting_list_customers:
            print('ci sono persone in attesa')
            message = f'Si sono liberati dei posti al ristorante {obj.restaurant.restaurant_name}'
            create_notification(customer, obj.restaurant, message)
        obj.delete()
        messages.success(request, message: 'Prenotazione cancellata con successo!')

    matteo pellacani
    def save_model(self, request, obj, form, change):
        super().save_model(request, obj, form, change)
        restaurant = obj.restaurant
        owner = obj.restaurant.owner
        message = (f'{restaurant.restaurant_name} prenotazione effettuata da {obj.customer.user.username} per il giorno: '
                   f'{obj.res_datetime}')
        create_notification(owner, obj.restaurant, message)
```

Il sistema di notifica viene gestito anche in base alle modifiche effettuate dall'admin.

POPOLAZIONE DATABASE

UTENTI

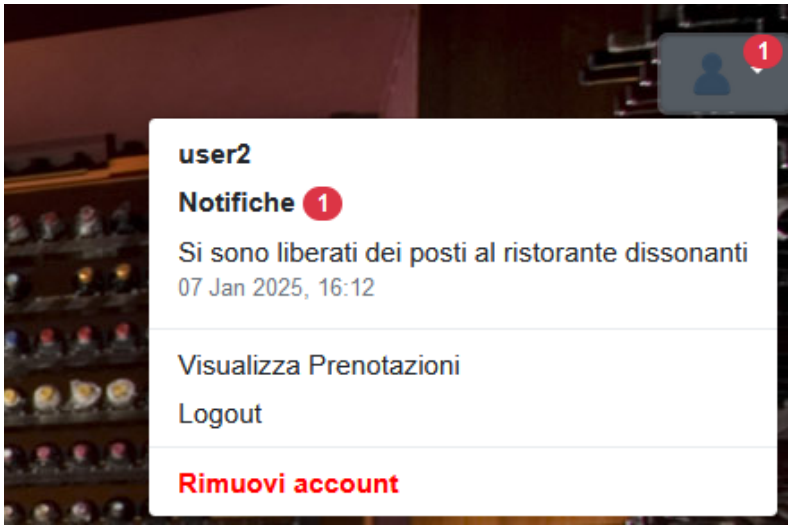
```
username,is_customer,is_owner,password,short_bio
user1,True,False,securepassword1,"Mi piace cucinare"
user2,True,False,securepassword2,"Amo viaggiare"
user3,True,False,securepassword3,"mi piace recensire ristoranti"
user4,True,False,securepassword4,"mi piace il buon cibo"
user5,True,False,securepassword5,"mi piace cucinare"
owner1,False,True,ownerpassword1
owner2,False,True,ownerpassword2
owner3,False,True,ownerpassword3
owner4,False,True,ownerpassword4
owner5,False,True,ownerpassword5
```

```
owner6,False,True,ownerpassword6
```

RISTORANTI

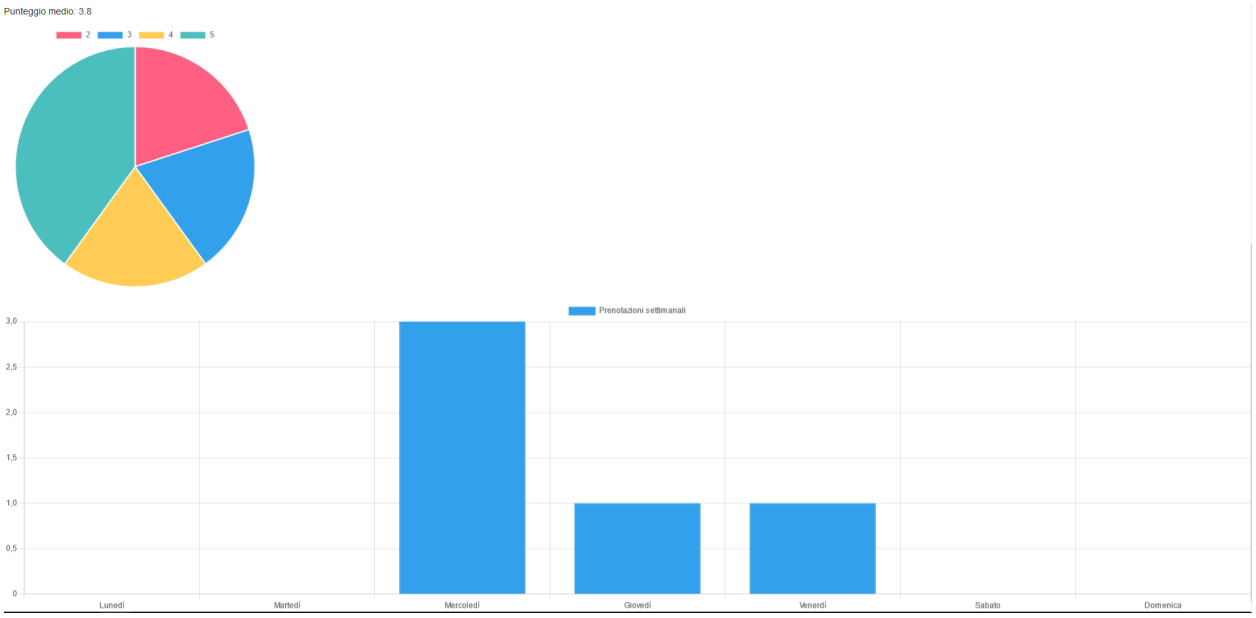
```
owner_id,restaurant_name,total_seats,city,start_lunch,end_lunch,start_dinner,end_dinner,address,imag
e,tags
15,crudo modena,40,Modena,11:00,13:00,19:00,23:00,via emilia est 953 modena,img\crudo
modena.jpg,pizza carne
15,osteria francescana,20,Modena,11:00,13:00,19:00,23:00,via stella 22
modena,img\osteria_francescana.jpg,carne pesce vegetariano
15,dissonanti,20,Modena,11:00,13:00,19:00,23:00,via jacopo berengario 110
modena,img\dissonanti.jpg,carne pesce pizza vegetariano
16,il mattatoio,20,Modena,11:00,13:00,19:00,23:00,via jacopo berengario 86
modena,img\mattatoio.jpg,carne
16,il pesciaio,20,Modena,11:00,13:00,19:00,23:00,str barchetta 280 modena,img\pesciaio.jpg,pesce
17,pizzikotto,20,Carpi,11:00,13:00,19:00,23:00,via aldo moro 22 carpi,img\pizzicotto.jpg,pizza carne
17,trattoria cognento,20,Carpi,11:00,13:00,19:00,23:00,via delle nazioni unite
carpi,img\cognento.jpg,carne pesce pizza vegetariano
18,bla bla,20,Carpi,11:00,13:00,19:00,23:00,via guastalla 66 carpi,img\blabla.jpg,carne pesce
18,acquapazza,20,Carpi,11:00,13:00,19:00,23:00,via gruppo 7 carpi,img\acquapazza.jpg,pesce
18,giustospirito,20,Carpi,11:00,13:00,19:00,23:00,via delle rezdore carpi,img\giustospirito.jpg,carne
pizza
```

FUNZIONE NOTIFICA



In questo esempio lo user2 aveva tentato una prenotazione presso il ristorante dissonanti, il ristorante essendo pieno nella fascia oraria scelta da user2, aveva notificato l'utente che non vi erano posti disponibili. L'utente si è quindi messo in lista di attesa, e alla successiva disdetta di una prenotazione per quella fascia oraria, lo user2 è stato notificato che si sono liberati dei posti per il ristorante per cui stava cercando di prenotare. Al click sulla notifica, essa scompare e lo user 2 viene rimosso dalla lista di attesa

ESEMPIO GRAFICI PER PROPRIETARIO



In questo caso il proprietario owner2, visualizza un grafico a torta corrispondente alla qualità delle recensioni, e settimanalmente può verificare il numero di prenotazioni per appunto quella settimana

TEST EFFETTUATI

alla base dei test effettuati vi è una forzatura per simulare un utente loggato

```
self.client.force_login(self.test_user)
```

I test sono stati effettuati utilizzando la libreria UNITEST presente in django, e sono state testate le seguenti funzionalità:

Per evitare un flusso di codice esagerato, mostro una parte di test relativa alla parte di prenotazione

Prenotazione:
Si verifica che una prenotazione venga effettuata con successo, e che venga effettivamente creata, inoltre che si venga reindirizzati correttamente come ci si aspetta

```
response = self.client.post(reverse('book:reservation', args=[self.restaurant.id]),
form_data)
self.assertEqual(response.status_code, 302)
self.assertTrue(Reservation.objects.filter(
  customer=self.customer,
  restaurant=self.restaurant,
  seats=2,
  res_datetime=datetime(2025, 1, 18, 18, 0)
).exists())
```

è stato testato il caso in cui venga effettuata una prenotazione sbagliat, e che venga visualizzato il relativo messaggio di errore.

Si è testato il caso in cui si vada a prenotare per un numero non valido, e quello che ci si deve aspettare è appunto una render e un messaggio di errore
Si è testato la corretta aggiunta alla waiting list nel caso in cui un utente voglia mettersi in lista di attesa a seguito di una prenotazione senza posti disponibili.

Recensioni
Si verifica che la recensione venga creata con successo e che non si possa effettuare una doppia recensione per uno stesso ristorante

Account
verifica la corretta creazione di un account, e i casi in cui si loggi con credenziali sbagliate, oppure che si vada ad accedere ad una pagina a cui non posso accedere, poichè non ho i permessi (vedi esempio)

```
def test_owner_wrong_access_infos(self):
  client = Client()
  client.login(username='test_user', password='test_password')
  response = client.get(reverse('book:watch_infos', kwargs={'restaurant':
self.restaurant.id}))
  self.assertEqual(response.status_code, 302)
  self.assertRedirects(response, '/login/?next=/book/infos/1/infos')
```

In questo caso un utente cerca di accedere ad una pagina riservata per i proprietari e quindi viene reindirizzato alla pagina di login.

PROBLEMI BUG RISCONTRATI

-problema presente nel test di verifica della lista di attesa, nonostante l'utente venga effettivamente aggiunto alla lista di attesa, il test non riesce a rilevare il reindirizzamento alla homepage

```
#self.assertEqual(response.status_code, 302)
```

-bug grafico inerente alla icona utente, che nella pagina di login è stato rimosso poichè non funzionante