



FIELD HUB

Piattaforma per la prenotazione di campi



11 LUGLIO 2024

ALBERTO NUZZACI – M. 165351
Esame di Tecnologie Web

Sommario

Introduzione	1
Abstract.....	1
Requisiti	1
Database	3
E/R Diagram.....	3
Parte degli Utenti.....	3
Parte dei Campi, Prenotazioni, Recensioni e Strutture	4
Implementazione.....	5
Tecnologie utilizzate	8
Frontend	8
jQuery & JS	8
Bootstrap	9
Backend	9
Organizzazione del codice	10
Views applicazione <i>core</i>	10
Views applicazione <i>users</i>	11
Recommendation System.....	12
Funzionamento.....	12
Formula.....	13
Unit Testing	14
Views	14
Models.....	14
Interfaccia	15
Parte del risultato	16
.....	16

Introduzione

Abstract

Applicazione Django destinata alla gestione di strutture sportive con campi sportivi associati. La piattaforma consente la **registrazione e la gestione di strutture sportive, campi sportivi con servizi associati, prenotazioni** in date e ore specifiche e, infine, **recensioni**; il tutto organizzato in modo differenziato per utenti utilizzatori dei campi e proprietari delle strutture.

Implementazione di **registrazione e accesso differenziato** da parte dei vari utenti.

Implementazione della **verifica dell'autenticità delle strutture** poiché chiunque può registrarsi come struttura senza avere permessi speciali.

Presenza di un **recommendation system** reviews-based che, basandosi sulle valutazioni degli utenti, riesce a fornire suggerimenti personalizzati.

Requisiti

Nel dettaglio, i requisiti dell'applicazione includono:

- **Visualizzazione, gestione e prenotazione dei Campi**
 - *Anonimi*: possono visualizzare i campi esistenti senza la possibilità di visualizzarne i dettagli, la disponibilità e, di conseguenza, effettuare prenotazioni.
 - *Utenti Registrati*: una volta registrati con i propri dati personali, possono cercare, visualizzare e prenotare i campi sportivi. Durante la visualizzazione dei dettagli di un certo campo sportivo, selezionato tra i risultati ottenuti dopo la ricerca, sarà presente:
 - Una foto del campo sportivo
 - Informazioni del campo: costo, copertura e servizi offerti.
 - Un pannello dedicato alla prenotazione
 - Informazioni sulla struttura del campo: descrizione, lista di collegamenti agli altri campi della struttura, contatti e indirizzo.
 - Possibili suggerimenti: *"Agli utenti a cui è piaciuto questo, è piaciuto anche..."*
 - Recensioni lasciate dagli utenti organizzate in più pagine.

Gli utenti registrati possono anche visualizzare le proprie prenotazioni divise in prenotazioni passate e future.

- *Proprietari di Strutture*: durante la registrazione non vengono richiesti solo i dati personali del proprietario ma anche i dati della struttura di sua competenza. Una volta registrati, i proprietari possono inserire all'interno della loro struttura dei campi selezionandone le caratteristiche e caricando una foto. I proprietari possono visualizzare i dettagli dei propri campi, le loro recensioni (cliccando sulla valutazione media associata ad ogni campo) e eliminarli una volta inseriti. I proprietari possono inoltre visualizzare e gestire le prenotazioni dei loro campi.
- **Funzionalità di Ricerca**
 - Gli utenti *anonimi* e gli *utenti registrati* possono ricercare campi personalizzando:
 - i. **Filtri**: è possibile ottenere dei risultati filtrando per tipo di sport (calcio a 7, calcio a 5, padel, tennis, ecc.), luogo (città della struttura) e copertura del campo (coperto oppure scoperto).

- ii. **Ordinamento:** è possibile ordinare i risultati ottenuti basandosi sulle recensioni, ossia voto medio delle recensioni di un campo, oppure in base al prezzo; entrambi in modo crescente oppure decrescente.
(di *default* i risultati vengono ordinati per recensioni decrescente)

- **Autenticità delle strutture**

L'autenticità delle strutture viene gestita tramite un valore booleano associato alla struttura: 0 (*default*) – non verificata, 1 – verificata.

- *Proprietari di Strutture:* una volta avvenuta la registrazione personale e, annessa, quella della struttura di competenza, di *default* la struttura sarà in stato *non verificata*. È presente nella sezione profilo una sezione che segnala lo status della verifica della struttura e, tramite un collegamento, è possibile richiedere una verifica della propria struttura mandando una mail all'*admin*.
- *Utenti Registrati & Anonimi:* i campi delle strutture non verificate verranno segnalati tramite un simbolo di warning (Δ) durante la visualizzazione dei risultati ottenuti a seguito della ricerca.
- *Utenti Registrati:* solo gli utenti registrati invece, una volta selezionato il campo di cui si desidera visualizzarne i dettagli e, eventualmente, procedere con la prenotazione, verranno avvisati tramite un pop-up di warning se la struttura non è ancora stata verificata. La prenotazione potrà comunque essere effettuata. Il concetto di “*possibilità*” di prenotazione nonostante la mancata verifica è stato pensato per poter essere applicato in un ipotetico scenario reale: lo staff del sito web si riserva da eventuali problemi con strutture non verificate.
- *Admin:* solo l'*admin* potrà procedere con la verifica vera e propria delle strutture. Una volta effettuato il login tramite il pannello *admin*, sempre accessibile tramite un collegamento presente nel *footer*, sarà presente, all'interno della tabella strutture, un'azione “*verifica le strutture selezionate*” che procederà ad autenticare le strutture selezionate. Gli utenti registrati non visualizzeranno più alcun segnale di warning nei campi associati alle strutture verificate.

- **Gestione delle prenotazioni**

- *Utenti Registrati:* durante la visualizzazione dei dettagli di un campo, un utente potrà procedere a prenotarlo selezionando la data con un calendario e l'ora con un menù a tendina; verranno mostrate solo le ore non prenotate per la data selezionata nel calendario. Alla conferma, verrà mostrata una pagina che segnala che la prenotazione è avvenuta con successo seguita da un breve *recap* dei dati di essa. Ogni utente registrato ha a sua disposizione un pannello dedicato alla gestione delle proprie prenotazioni. In particolare, le prenotazioni vengono divise in prenotazioni passate e prenotazioni future. Le prenotazioni future possono essere eliminate. Le prenotazioni passate, invece, permettono all'utente di lasciare una recensione al campo associato a quella prenotazione; ogni utente può recensire un campo solo una volta (anche se lo ha prenotato più volte) e solo dopo averne usufruito (la data della prenotazione è passata). Vincoli a livello di *models* impediscono il salvataggio di recensioni che non rispettano questi criteri.
- *Proprietari di Strutture:* ogni proprietario potrà vedere la lista delle prenotazioni dei suoi campi ordinate in modo cronologico. Nello specifico, potrà eliminare solo le prenotazioni con data futura. È inoltre presente un tasto “*esporta prenotazioni*” che permette all'utente proprietario, mediante l'apertura di un *modal*, di esportare le prenotazioni in un certo range di date. Questa implementazione è stata introdotta al fine di offrire una soluzione pratica e conveniente per la gestione delle prenotazioni, rendendo più agevole l'accesso in un ipotetico contesto operativo reale e quotidiano.

- **Registrazione e gestione del profilo**

- **Registrazione:** la registrazione avviene in maniera differenziata per *Utenti* e *Proprietari di Strutture*. Prima che avvenga la registrazione vera e propria viene chiesto all'utente se si tratta di un *Utente* o di un *Proprietario di una Struttura*; seguirà poi, se si tratta di un *Utente*, una pagina dedicata al solo inserimento dei dati personali, invece, se si tratta di un *Proprietario di una Struttura*, seguirà una pagina dedicata sia all'inserimento dei dati personali che all'inserimento dei dati della *Struttura*.
- **Gestione:** la gestione del profilo per entrambi i tipi di utenti avviene nel modo medesimo. È presente infatti un pannello *Profilo* che permetterà la modifica dei dati: solo personali se si tratta di un *Utente*, anche della struttura se si tratta di un *Proprietario di una Struttura*. In particolare, dopo la registrazione tutti i tipi di utenti verranno registrati con una foto profilo di default che potrà essere cambiata proprio in questo pannello. È inoltre presente la funzionalità di cambio password.
- **Test**
 - Presenti test nell'applicazione *core* che mirano a garantire il corretto funzionamento delle logiche dell'applicazione.

Database

Il DBMS scelto è stato *SQLite* poiché:

- Non richiede installazioni o configurazioni separate poiché integrato in Python.
- Non è necessario un server poiché i dati sono memorizzati in un singolo file.
- Efficienza per carichi leggeri/medi

Queste tre combinazioni garantiscono rapidità di avvio e una sufficiente performance per una fase iniziale di un progetto Django.

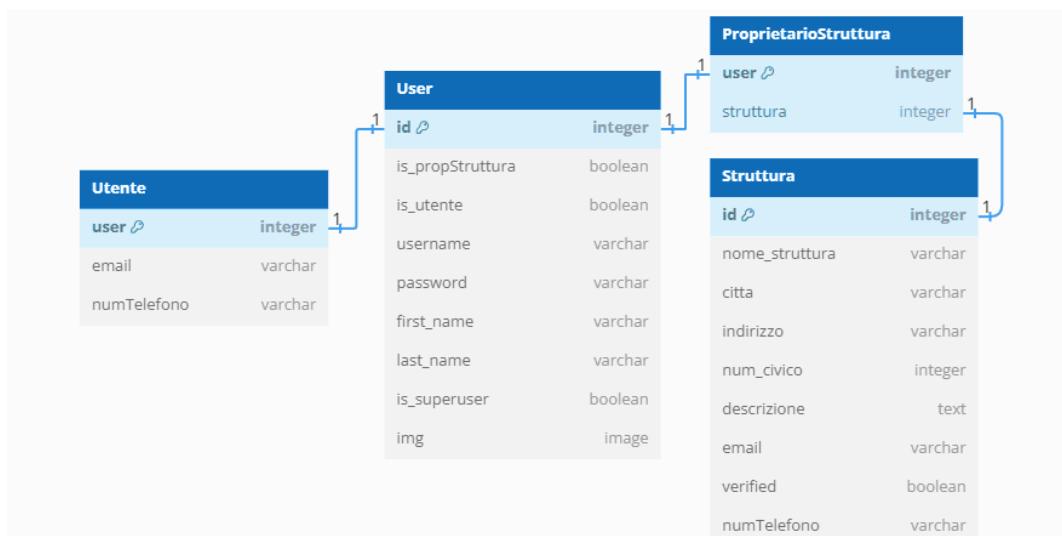
E/R Diagram

Lo schema E/R viene mostrato nelle due figure successive ([figura 1](#) e [figura 2](#)).

Diviso in due figure solo per comodità rappresentative (la tabella *strutture* e *utenti* fanno riferimento alle stesse in entrambe le figure).

Parte degli Utenti

Questa parte del DB mostrato nella [figura 1](#) è la parte dedicata alla gestione degli utenti.

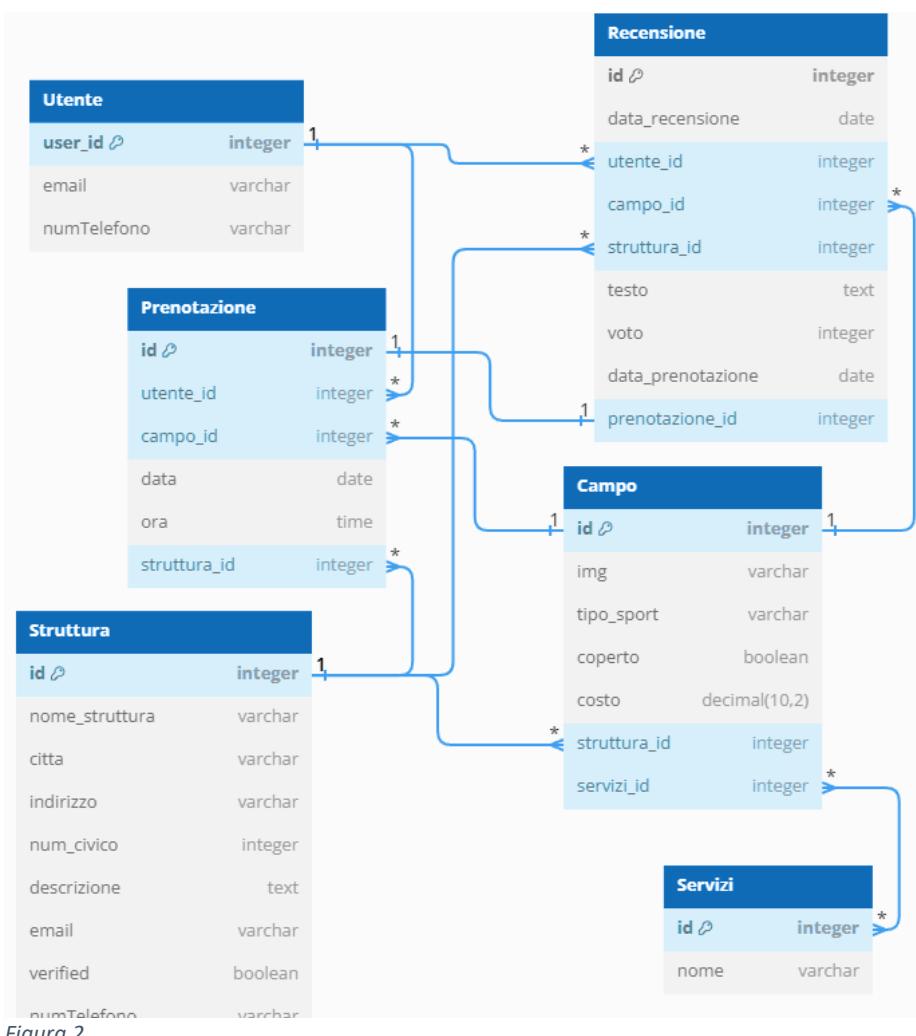


In particolare, la tabella:

- **Users:** estende il modello `AbstractUser` di Django per includere campi aggiuntivi. I campi aggiuntivi `is_propStruttura` e `is_utente` sono booleani che indicano rispettivamente se l'utente è un proprietario di una struttura o un normale utente, mentre `img` è dedicato all'immagine profilo dell'utente. L'immagine alla creazione sarà imposta con una di default modificabile poi una volta avvenuto il *login*. Gli altri campi fanno invece riferimento a quelli del modello `AbstractUser` di Django, per avere una migliore visualizzazione sono stati riportati solo quelli usati. La tabella **User** ha un riferimento *uno-a-uno* sia con la tabella **ProprietarioStruttura** che con la tabella **Utente**; questo tipo di relazione garantisce che non ci sia un **Proprietariostruttura** con lo stesso username di un **Utente**.
- **Utente:** rappresenta un utente normale del sistema. Ha una relazione *uno-a-uno* con la tabella `User`, quindi ogni **Utente** corrisponde esattamente a un **User**. Contiene campi aggiuntivi specifici per l'utente, come l'`email` e il numero di telefono (`numTelefono`).
- **Proprietariostruttura:** rappresenta un proprietario di una struttura. Ha una relazione *uno-a-uno* con la tabella `User`, simile a **Utente**. Contiene un riferimento a una `Struttura` indicando che ogni proprietario gestisce una specifica struttura.

Parte dei Campi, Prenotazioni, Recensioni e Strutture

Questa parte del DB mostrata nella [figura 2](#) è impegnata a gestire tutto ciò che riguarda la gestione delle prenotazioni, dei campi con strutture annesse e, infine, delle recensioni.



In particolare, la tabella:

- **Struttura:** rappresenta un luogo o un impianto sportivo e contiene i seguenti campi: nome_struttura, citta, indirizzo, num_civico, descrizione, email, verified, numTelefono. Ha una relazione *uno-a-uno* con propStruttura indicando che c'è un unico proprietario per ogni struttura. Ha una relazione *uno-a-molti* con la tabella **Campo**: ogni struttura ha n campi.
- **Campo:** rappresenta i campi sportivi all'interno delle strutture e contiene un'img del campo, tipo_sport (ad esempio Tennis, Calcio a 5, Padel, ecc.), un campo booleano coperto che indica se il campo è coperto, il costo, e i **Servizi**. I servizi non sono altro che una relazione *molti-a-molti* con la tabella **Servizi**. Questa tabella ha una relazione *molti-a-uno* con il modello **Struttura**, indicando a quale struttura appartiene il campo.
- **Prenotazione:** rappresenta una prenotazione di un campo da parte di un utente. Contiene il campo (relazione *molti-a-uno* con la tabella **Campo**), data della prenotazione, ora della prenotazione, utente (relazione *molti-a-uno* con la tabella **Utente**), struttura (relazione *molti-a-uno* con la tabella **Struttura**). Sono presenti vincoli a livello di database che impediscono:
 - Il salvataggio di una prenotazione per conto di una struttura; l'utente che richiede una prenotazione non deve essere un proprietario di una struttura.
 - La prenotazione non venga effettuata in un'ora intera compresa tra le 10:00 e le 21:00
 - Il salvataggio di una prenotazione di un campo in un'ora già occupata: è presente un vincolo unique_together che assicura che non ci siano due entry nella tabella **Prenotazione** con la stessa combinazione di valori per i campi campo, data e ora.
- **Recensione:** rappresenta una recensione associata a una certa prenotazione. Contiene la data_recensione ossia la data in cui è stata scritta la recensione, l'utente (relazione *molti-a-uno* con la tabella **utente**), il testo vero e proprio della recensione, il voto, la data_prenotazione ossia la data della prenotazione associata e la prenotazione vera e propria (relazione *uno-a-uno* con la tabella **prenotazione**). Sono presenti vincoli a livello di database che impediscono che:
 - Il salvataggio di una recensione avvenga per conto di una struttura, l'utente che richiede una recensione non deve essere il proprietario di una struttura.
 - Un utente recensisca per altri utenti; l'utente che richiede il salvataggio di una recensione deve essere lo stesso associato alla prenotazione
 - La data della prenotazione non sia passata rispetto alla recensione; si può recensire un campo solo dopo l'uso ossia dopo che la data della prenotazione è passata.
 - Il salvataggio di un voto con valore diverso tra gli interi compresi da 1 a 5.
 - Un utente possa lasciare più di una recensione per lo stesso campo. È presente infatti un vincolo unique_together che assicura che la combinazione dei campi utente e campo sia unica all'interno della tabella.

Implementazione

L'implementazione delle diverse tabelle, nonché modelli di Django, è stata suddivisa in due applicazioni:

- **Core**
L'applicazione **core** gestisce le entità principali del sistema, inclusi campi sportivi, servizi, prenotazioni, recensioni e strutture. Utilizza relazioni tra i modelli descritte precedentemente per organizzare e collegare queste entità in modo efficiente.

- **Users**

L'applicazione *users* gestisce le informazioni degli utenti, distinguendo tra utenti normali e proprietari delle strutture fornendo quindi supporto per un'autenticazione e profili utente personalizzati. In particolare, all'interno del file `settings.py`, si può trovare

```
AUTH_USER_MODEL = 'users.User'
```

che serve per specificare che il modello utente personalizzato, definito all'interno di questa applicazione, dovrà essere utilizzato come modello utente predefinito per l'autenticazione e la gestione degli utenti all'interno dell'intero progetto.

Segue una breve panoramica sull'implementazione vera e propria, con particolare attenzione alle relazioni.

Applicazione Core

```
class Servizio(models.Model):
    SERVIZIO_CHOICES = [
        ('docce', 'Docce'),
        ('piscina', 'Piscina'),
        ('spogliatoi', 'Spogliatoi'),
        ('armadietti', 'Armadietti'),
        ('illuminazione', 'Illuminazione notturna'),
        ('bar', 'Bar'),
        ('parcheggio', 'Parcheggio gratuito'),
        ('noleggiorachette', 'Noleggio racchette'),
        ('casacche', 'Casacche in prestito')
    ]
    nome = models.CharField(max_length=50,
                           choices=SERVIZIO_CHOICES,
                           unique=True)
    ...
    ...

class Campo(models.Model):
    TIPO_SPORT_CHOICES = [
        ('tennis', 'Tennis'),
        ('calcio5', 'Calcio a 5'),
        ('calcio7', 'Calcio a 7'),
        ('padel', 'Padel'),
        ('beachvolley', 'Beach Volley'),
        ('pallavolo', 'Pallavolo')
    ]
    tipo_sport = models.CharField(max_length=50,
                                 choices=TIPO_SPORT_CHOICES)
    servizi = models.ManyToManyField(Servizio)
    struttura = models.ForeignKey(Struttura,
                                  on_delete=models.CASCADE)
    ...
    ...
```

In particolare la relazione *multi-a-multi* è realizzata mediante il tipo di campo `models.ManyToManyField`, mentre la relazione *multi-a-uno* è realizzata tramite `models.ForeignKey`. Si noti che `TIPO_SPORT_CHOICES` e `SERVIZIO_CHOICES` corrispondono a una lista di valori che potranno assumere i campi `nome` (nel modello *Servizio*) e `tipo_sport` (nel modello *Campo*). Questo risulta particolarmente utile per diverse ragioni:

- **Interfaccia Utente:**
 - **Dropdown Menù & Checkbox:** queste liste di valori possono essere utilizzate per popolare i *dropdown menù* e *checkbox* molto facilmente nei form del front-end.
- **Scalabilità del Codice:**
 - **Facilità di Aggiornamento:** aggiungere un nuovo tipo di sport o un nuovo servizio è semplice e immediato. Basta aggiungere un nuovo elemento alla lista `TIPO_SPORT_CHOICES` o `SERVIZIO_CHOICES`, e il sistema è subito aggiornato per gestire le nuove opzioni.

```

class Prenotazione(models.Model):
    utente = models.ForeignKey('users.Utente',
        on_delete=models.CASCADE)
    campo = models.ForeignKey(Campo,
        on_delete=models.CASCADE)
    struttura = models.ForeignKey(Struttura,
        on_delete=models.CASCADE)
    ...
    class Meta:
        unique_together = ('campo', 'data', 'ora')

    def save(self, *args, **kwargs):
        if len(str(self.ora)) > 5:
            raise ValueError("L'ora deve essere nel formato %H:%M ")
        ora, minuti = self.ora.split(':')
        if not self.utente.user.is_utente:
            raise ValueError(...)
        if int(minuti) != 0:
            raise ValueError(...)
        if int(ora) < 10 or int(ora) > 21:
            raise ValueError(...)
        super().save(*args, **kwargs)

class Recensione(models.Model):
    utente = models.ForeignKey('users.Utente',
        on_delete=models.CASCADE )
    campo = models.ForeignKey('Campo',
        on_delete=models.CASCADE )
    struttura = models.ForeignKey('Struttura',
        on_delete=models.CASCADE )
    voto = models.IntegerField(
        validators=[MinValueValidator(1),
                    MaxValueValidator(5)])
    prenotazione = models.OneToOneField('Prenotazione',
        on_delete=models.CASCADE)
    ...
    class Meta:
        unique_together = ('utente', 'campo')

    def save(self, *args, **kwargs):
        if not self.utente.user.is_utente:
            raise ValueError(...)
        if self.data_recensione < self.data_prenotazione:
            raise ValueError(...)
        if self.prenotazione.utente != self.utente:
            raise ValueError(...)
        super().save(*args, **kwargs)

```

La relazione *uno-a-uno* è realizzata mediante il campo `models.OneToOneField`.

Si noti che i vincoli spiegati precedentemente (per il modello [Prenotazione](#) e il modello [Recensione](#)) sono stati realizzati mediante l'*override* del metodo `save`; il metodo verrà chiamato tutte le volte che si tenta di inserire una entry nella tabella corrispondente.

Invece, i vincoli `unique_together`, sono stati definiti tramite la classe interna `Meta` dei modelli Django. Si tratta di una classe speciale che permette di definire metadati fornendo informazioni aggiuntive e ulteriori personalizzazioni sul comportamento del modello, senza influenzare direttamente i campi del modello stesso.

L'eliminazione è di una entry delle varie tabelle è gestita in maniera opportuna mediante `on_delete=models.CASCADE` (es. all'eliminazione di un campo vengono eliminate tutte le prenotazioni corrispondenti).

Applicazione Users

Modello User che eredita da `AbstractUser` offerto da Django.

Alla registrazione viene assegnata un'immagine di default all'utente, modificabile inseguito

```

class User(AbstractUser):
    is_propStruttura = models.BooleanField(default=False)
    is_utente = models.BooleanField(default=False)
    img = models.ImageField(upload_to='static/img/users_img/profile_pic',
                           default='static/img/default_img/no_img_profile.jpg')

```

```

class Utente(models.Model):
    user = models.OneToOneField(User,
        on_delete=models.CASCADE,
        primary_key=True)
    email = models.CharField(max_length=100)
    numTelefono = models.CharField(max_length=13)

class ProprietarioStruttura(models.Model):
    user = models.OneToOneField(User,
        on_delete=models.CASCADE,
        primary_key=True)
    struttura = models.OneToOneField(Struttura,
        on_delete=models.CASCADE)

```

La chiave primaria per entrambi i modelli è realizzata mediante una relazione *uno-a-uno* col modello **User**. Per ognuno dei due utenti vengono aggiunti i relativi dati.

Si noti che ogni **Proprietario** potrà gestire solo una struttura.

Tecnologie utilizzate

Frontend

Le tecnologie utilizzate lato front-end includono *JavaScript*, *HTML* e *CSS*. In particolare:

- **Bootstrap 5.3**: amplia il *CSS* fornendo una vasta gamma di componenti predefiniti e di stile. Include una serie di classi *CSS* pronte all'uso che rendono più facile la creazione di layout reattivi e ben progettati senza dover scrivere molto codice *CSS* personalizzato.
- **jQuery**: amplia *JavaScript* semplificando la manipolazione del **DOM**, la gestione degli eventi, le animazioni e le richieste **AJAX**. *jQuery* offre una sintassi più semplice e concisa rispetto al *JavaScript* puro, facilitando operazioni comuni che altrimenti richiederebbero più codice.
- **Font Awesome**: libreria che amplia ulteriormente il *CSS* già offerto da bootstrap. Permette infatti di incorporare facilmente diverse icone (es. stelle per le recensioni, icone per i pulsanti ecc.) al fine di migliorare l'estetica del sito. Il tutto volto a garantire una migliore UX.

Tutte queste tecnologie sono state ampiamente utilizzate nello sviluppo e scrittura dei vari template che compongono l'applicazione.

jQuery & JS

- **Modali di conferma azioni**

Utilizzo ampio *JavaScript* e *jQuery* per implementare modali che notificano all'utente l'avvenuto successo di specifiche azioni come il login o l'eliminazione di un elemento (es. Campo o Prenotazione). In particolare, sfruttando i modali offerti da *Bootstrap* è stato possibile fornire un feedback visivo immediato agli utenti dopo la redirezione a una nuova pagina. Questo assicura che gli utenti siano sempre consapevoli e notificati del risultato delle loro azioni.

- **Modali di ingrandimento immagini**

Utilizzo di *JS* e *jQuery* per l'ingrandimento di immagini dei campi (lato *Struttura*) tramite *modal*.

- **Interazione dinamica con AJAX**

Utilizzati per facilitare l'interazione dinamica con l'interfaccia utente e migliorare la User Experience. È stata infatti utilizzata la tecnologia **AJAX** che, con il supporto della libreria *moment.js* per convertire la data del *datepicker* Bootstrap in un formato opportuno, permette di inviare una richiesta *AJAX* al server per ottenere le ore disponibili di un certo campo nella data selezionata. L'aggiornamento dinamico del menù dropdown delle ore disponibili crea un approccio dove viene migliorata l'interattività, la reattività della pagina e, di conseguenza, viene fornito un feedback immediato all'utente.

- **Tasti di eliminazione elementi e aggiunta recensioni**

Un terzo utilizzo viene fatto per creare tasti di eliminazione di elementi (prenotazioni lato utente/struttura e campi) in modo generale e per la creazione di recensioni con dati aggiuntivi tramite modali. Ad esempio, quando l'utente clicca su un pulsante di eliminazione, viene mostrato un modale di conferma che riporta i dettagli dell'elemento da eliminare. Allo stesso modo, per aggiungere recensioni, un modale permette all'utente di inserire il voto e il testo della recensione, migliorando così l'interazione e garantendo una gestione intuitiva delle operazioni di eliminazione e recensione direttamente dall'interfaccia utente.

- **Validazione dati**

Facendo riferimento all'esportazione di prenotazioni lato Proprietario Struttura, è stato utilizzato *JavaScript* per implementare la validazione dei dati inseriti dall'utente nel modale aperto. Una funzione *JavaScript* infatti controlla che la data di fine sia successiva alla data di inizio, mostrando un messaggio di errore se la validazione fallisce.

Bootstrap

- **Layout**

Gestione esclusiva del Layout effettuata con *Bootstrap*. È infatti particolarmente semplice, intuitivo ed efficace posizionare in modo arbitrario gli elementi sulla pagina. Bootstrap offre infatti due potenti sistemi di layout, *Flex* e *Grid*. *Flex* utilizza il modulo *CSS Flexbox* per creare layout flessibili e reattivi, mentre *Grid* sfrutta un sistema a 12 colonne che consente di organizzare i contenuti in righe e colonne, garantendo un design coerente, facilmente scalabile su diversi dispositivi il tutto gestito con poche righe di codice e una sintassi molto intuitiva.

- **Datepicker**

Utilizzo del componente *Datepicker* offerto da *Bootstrap* (aggiunta una libreria per la traduzione in italiano dei mesi). Questo strumento consente agli utenti di selezionare facilmente una data da un calendario interattivo, migliorando l'usabilità del form. Il *Datepicker* è stato configurato per utilizzare il formato di data *dd/mm/yyyy*, chiudersi automaticamente dopo la selezione della data e supportare la localizzazione in lingua italiana. Questa soluzione permette di inserire date in modo intuitivo e previene errori di formattazione, garantendo che i dati raccolti siano coerenti e validi.

- **Aspetto estetico generale**

Avendo classi CSS predefinite, è stato intuitivo creare una pagina che possa avere un aspetto più invitante per l'utente.

- **Pagine di errore personalizzate**

Utilizzato inoltre per creare, chiaramente con un supporto nel *backend*, pagine personalizzate d'errore. In particolare sono state create pagine di accesso proibito (403) e di risorsa non trovata (404) in modo tale che, nel caso in cui l'utente finisca in una pagina d'errore, possa autonomamente redirezionarsi verso una pagina corretta del sito web.

- **Forms**

I form sono stati interamente gestiti a livello grafico con *Bootstrap*. È stato deciso di creare form personalizzati piuttosto che renderizzarli con *crispy forms* poiché questo garantisce una personalizzazione molto più dettagliata. Scrivendo proprio codice HTML per i form permette un controllo completo su ogni singolo aspetto del layout e dello stile.

Backend

Il backend è formato da endpoint definiti dalle **Django Views**, nonché componenti del framework Django, che gestiscono la logica di elaborazione delle richieste HTTP e restituiscono le risposte fornite tramite il **Django Template Engine** sottoforma di *HTML*.

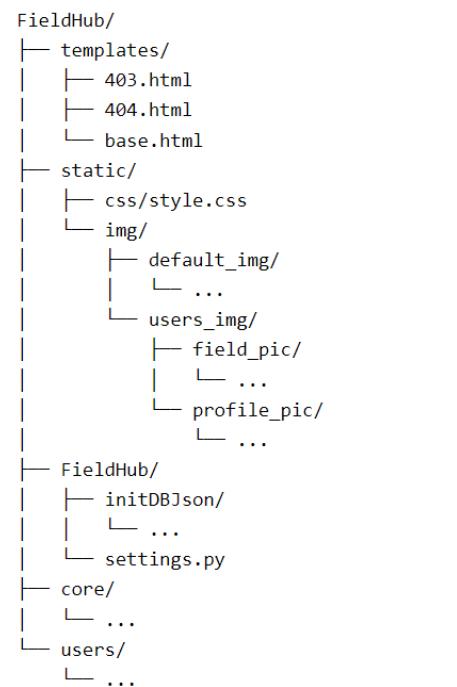
È previsto un endpoint **API RESTful** `/core/ore_libere/<int:id_campo>/YYYY-MM-DD/` che permette, solo agli utenti autenticati come *Utente*, di visualizzare sotto forma di JSON le ore libere di un determinato campo in una specifica data. Questo endpoint viene richiamato tramite *AJAX* dalla *DetailView* dedicata alla visualizzazione del campo.

Per la gestione dei permessi alle varie pagine è stata utilizzata la classe `LoginRequiredMixin` offerta da `django.contrib.auth.mixins`; sono state create due classi che, ereditando dall'ultima nominata, hanno permesso di creare due comportamenti personalizzati richiesti per *Utente* e *ProprietarioStruttura*. Queste due classi, insieme ad ulteriore logica implementata tramite il metodo `dispatch` nelle singole *ClassBasedViews*, hanno permesso di gestire in maniera istitutiva e comoda i diversi tipi di permessi e, quindi, nei casi necessari, negare l'accesso a determinate *views*.

Organizzazione del codice

Come accennato precedentemente il progetto è diviso in due applicazioni:

- **Users:** contiene la definizione dei tre modelli dedicati alla gestione degli utenti, implementa le viste e i template relativi alla registrazione, al login, al logout e alla gestione del profilo.
- **Core:** contiene la definizione di tutti i modelli principali, implementa le viste e i template relativi a tutte le altre schermate: *Home* (diversa per Utente e Strutture), *Gestione prenotazioni* (diversa per Utenti e Strutture), *Lista Campi*, *Visualizza Campo*, *Crea Campo*. Al suo interno è stato aggiunto un file extra rispetto a quelli generati da Django `recommendations.py` dove viene implementato la funzione per il Recommendation System.



All'interno di `./FieldHub/templates` è presente il template scheletro del progetto `base.html` e i due template per le pagine di errore personalizzate. Mentre in `./FieldHub/static` sono presenti le immagini opportunamente divise e il css. Più precisamente, `default_img` conterrà le immagini di default con cui verrà popolato inizialmente il database, mentre `users_img` conterrà le immagini caricate dagli utenti organizzate in due sotto-cartelle, una per i campi e l'altra per le immagini profilo.

Views applicazione core

Nome della View	Tipo di View	Dettagli	Anonimi	Proprietari Struttura	Utenti
<code>home_page</code>	<code>F.B. View</code>	In base al tipo di utente ridireziona alla <code>home_page</code> .	✓	✓	✓
<code>ListaCampiView</code>	<code>ListView</code>	Mostri i campi trovati a seguito di una ricerca.	✓	✗	✓
<code>CercaCampoView</code>	<code>ListView</code>	Permette di ricercare campi. Home page di anonimi e Utenti.	✓	✗	✓
<code>PrenotazioniUtenteView</code>	<code>TemplateView</code>	Mostra le prenotazioni di un certo utente divise in future e passate.	✗	✗	✓
<code>PrenotazioneConfermataView</code>	<code>TemplateView</code>	Mostra la conferma di una prenotazione.	✗	✗	✓

<code>DetailCampoView</code>	<code>ListView</code>	Mostra i dettagli di un campo.	✗	✗	✓
<code>DetailStrutturaView</code>	<code>DetailView</code>	Mostra i dettagli di una struttura.	✗	✗	✓
<code>ore_libere</code>	<code>F.B. View</code>	API RESTful endpoint accennato in precedenza .	✗	✗	✓
<code>salva_recensione</code>	<code>F.B. View</code>	Permette di aggiungere una recensione.	✗	✗	✓
<code>RecensioniCampoView</code>	<code>ListView</code>	View che permette di visualizzare le recensioni di un certo campo per le strutture.	✗	✓	✗
<code>PrenotazioniStrutturaView</code>	<code>TemplateView</code>	Visualizza tutte le prenotazioni di una certa struttura.	✗	✓	✗
<code>elimina_campo</code>	<code>F.B. View</code>	Permette lato struttura di eliminare un campo.	✗	✓	✗
<code>create_campo</code>	<code>F.B. View</code>	Permette lato struttura di eliminare un campo.	✗	✓	✗
<code>ListaCampiPerStrutturaView</code>	<code>ListView</code>	Elenca i campi di una certa struttura fornendo la possibilità di eliminarli. Home page di strutture.	✗	✓	✗
<code>esporta_prenotazioni</code>	<code>F.B. View</code>	Permette di esportare le prenotazioni in un file txt di una certa struttura.	✗	✓	✗
<code>elimina_prenotazione</code>	<code>F.B. View</code>	Permette di eliminare una prenotazione con comportamenti diversi lato struttura/utente.	✗	✓	✓

Views applicazione users

Nome della View	Tipo di View	Dettagli	Anonimi	Proprietari Struttura	Utenti
<code>register</code>	<code>F.B View</code>	View che mostra la pagina di scelta utente da registrare	✓	✗	✗
<code>login_request</code>	<code>F.B View</code>	View per effettuare il login.	✓	✗	✗
<code>PropStrutturaRegistrationView</code>	<code>CreateView</code>	Permette di registrare il proprietario di una struttura.	✓	✗	✗
<code>UtenteRegistrationView</code>	<code>CreateView</code>	Permette di registrare un utente.	✓	✗	✗
<code>propStrutturaUpdateView</code>	<code>F.B View</code>	Permette di modificare i dati del profilo di un proprietario di una struttura.	✗	✓	✗
<code>utenteUpdateView</code>	<code>F.B View</code>	Permette di modificare i dati di un Utente.	✗	✗	✓
<code>logout_request</code>	<code>F.B View</code>	Permette di eseguire un logout.	✗	✓	✓
<code>view_profile</code>	<code>F.B View</code>	Permette di visualizzare i dati del profilo.	✗	✓	✓
<code>update_profile</code>	<code>F.B View</code>	Redirige alla pagina di modifica del profilo a seconda del tipo di utente loggato.	✗	✓	✓
<code>cambia_password</code>	<code>F.B View</code>	Permette di cambiare la password.	✗	✓	✓

Recommendation System

Il sistema di raccomandazione è stato progettato per suggerire campi sportivi basandosi sulle recensioni degli utenti che hanno già recensito campi dello stesso tipo di quello oltre a quello che si sta visualizzando (stesso tipo di sport).

L'obiettivo principale è dare maggiore rilevanza alle opinioni degli utenti che hanno recensito positivamente il campo visualizzato; perciò lo “*score*”, in termini di possibili *recommended*, sarà alto nel momento in cui sia il campo visualizzato che, contemporaneamente, il possibile suggerito sono stati recensiti positivamente dal medesimo utente u . Questo tipo di approccio permette di proporre all'utente che sta visualizzando il campo in questione:

“Agli utenti a cui è piaciuto questo campo è piaciuto anche...”

Funzionamento

1. **Raccolta delle recensioni:** si raccolgono tutte le recensioni per il campo sportivo visualizzato x .
2. **Raccolta degli utenti:** si raccolgono tutti gli utenti $[u_1, u_2, \dots, u_n]$ delle recensioni del campo sportivo x .
3. **Raccolta dei campi:** si raccolgono tutti i campi dello stesso tipo (es. se sto visualizzando un campo per *calcio a 5* considererò solo i campi per *calcio a 5*) recensiti da $[u_1, u_2, \dots, u_n]$ escluso x . Chiaramente potranno esserci duplicati, poiché ipoteticamente u_1 e u_2 potrebbero non solo aver recensito x ma anche y .
4. **Calcolo dei voti ponderati:** fisso un utente u , scorro i campi che ha recensito (sempre escluso x), calcolo ogni voto come

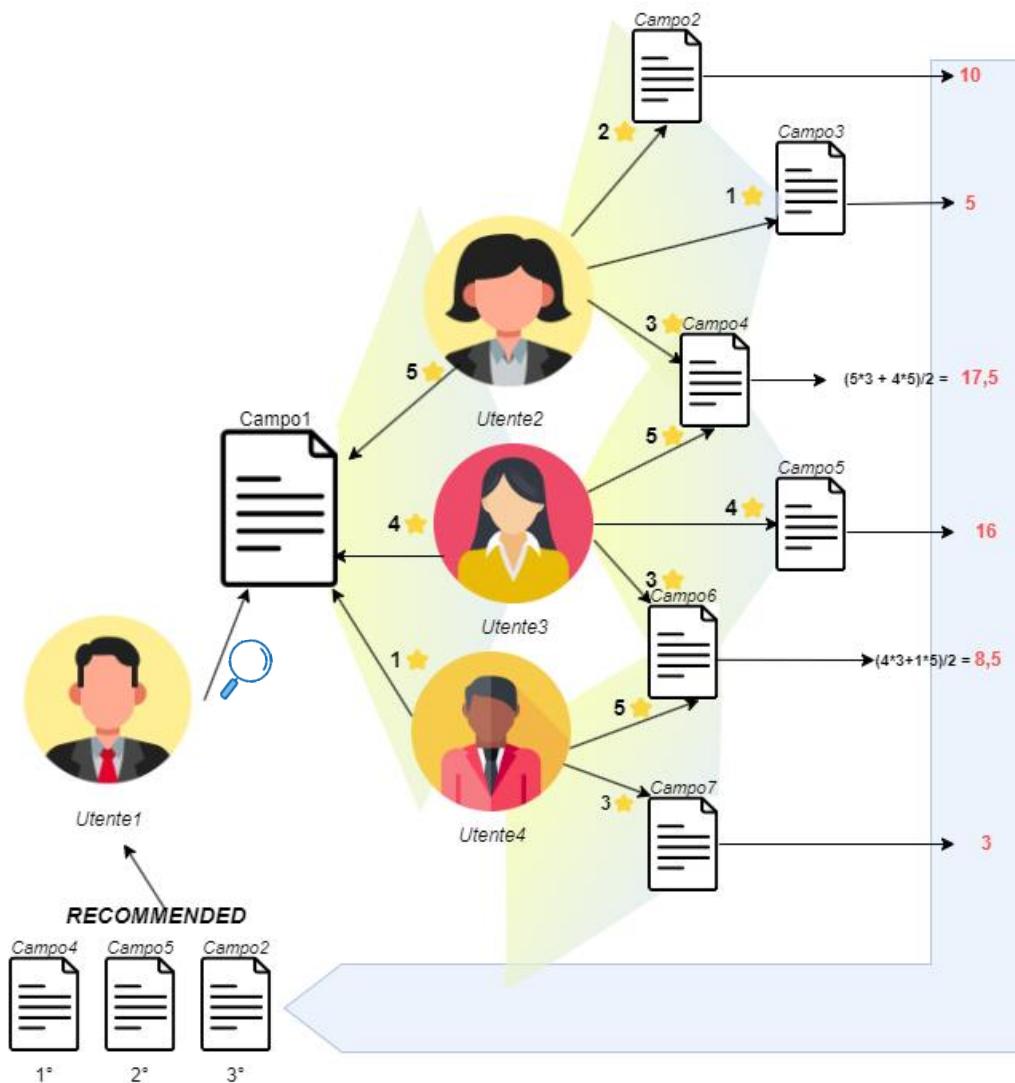
$$\text{voto di } x * \text{voto di } y$$

Dove:

- Voto di x : voto dato al campo visualizzato dall'utente u
 - Voto di y : voto dato al campo y , ossia un campo della lista, dall'utente u
5. **Calcolo delle medie:** fisso un campo, calcolo la media dei voti ponderati di ogni campo.
 6. **Generazione delle Raccomandazioni:** si selezionano gli n (di default 3) campi sportivi con le medie più alte e verranno suggeriti all'utente.

Di default è previsto che vengano ritornati solo considerati i campi dello stesso tipo; tutta via è presente un valore booleano che, potendo essere cambiato facilmente, permette di ottenere un *recommendation system* senza fare delle esclusioni per tipo

Per una visione più chiara è stato previsto uno **schema riassuntivo**.



Formula

Una volta trovati tutti i campi dello stesso tipo recensiti dagli utenti che hanno anche recensito il campo che si sta visualizzando, l'algoritmo usato per il calcolo dello score può essere riassunto con la seguente formula.

$$S(y) = \frac{\sum_{u \in U_x \cap U_y} V_x(u) * V_y(u)}{|U_x \cap U_y|}$$

Dove:

- $y \in [y_1, y_2 \dots y_n]$ ossia y appartiene alla lista dei campi trovati
- $S(y)$ è lo score del campo y
- $U_{x|y}$ è l'insieme degli utenti che hanno recensito il campo $x|y$.
- $V_{x|y}(u)$ è il voto dato dall'utente u al campo $x|y$.
- $|U_x \cap U_y|$ è il numero di utenti che hanno recensito sia x che y .

Unit Testing

Le funzionalità coperte dallo *unit testing* sono in tutto 12, 6 per i modelli e 6 per le views. Tutti i test risiedono nel file dedicato *FieldHub/core/tests.py* all'interno dell'applicazione principale.

Views

- **EliminaPrenotazioneViewTest**
 - `test_delete_just_own_prenotazioni`: verifica che un utente possa eliminare solo le proprie prenotazioni.
- **PrenotazioniUtenteViewTest**
 - `test_last_old_prenotazione_before_today`: verifica che l'ultima recensione dell'ultima pagina delle prenotazioni passate sia antecedente alla data attuale.
 - `test_no_prenotazione_nuovo_utente`: verifica che un nuovo utente non abbia alcuna prenotazione.
 - `test_review_button_disabled_for_reviewed_booking`: verifica che il pulsante per recensire una prenotazione già recensita sia disabilitato.
- **OreLibereTest**
 - `test_ore_libere_no_prenotazioni`: verifica che in un giorno senza prenotazioni vengano ritornate tutte le ore disponibili.
 - `test_ore_libere_con_prenotazioni`: verifica che in un giorno con prenotazioni vengano escluse le ore già prenotate.

Models

- **PrenotazioneModelTest**
 - `test_unique_together`: Verifica che ogni prenotazione sia univoca in termini di campo, data e ora.
 - `test_only_utente`: verifica che solo gli utenti comuni possano effettuare prenotazioni, escludendo i proprietari di strutture.
 - `test_only_oclock_between_10_21`: verifica che le prenotazioni possano essere effettuate solo in ore intere tra le 10:00 e le 21:00 comprese.
- **RecensioneModelTest**
 - `test_invalid_recensione_data`: verifica che sia possibile recensire solo prenotazioni con date passate.
 - `test_unique_recensione_campo`: verifica che un utente possa recensire un determinato campo una sola volta.
 - `test_recensione_for_other_user`: verifica che un utente possa recensire solo per sé stesso e non per altri utenti.

Interfaccia

L'interfaccia è diversamente gestita in base al tipo di utente loggato. Le pagine del sito sono state pensate per mantenere una *navbar* e un *footer* sempre visibili, rispettivamente per spostarsi tra le varie pagine e accedere al pannello admin.

La *navbar* è organizzata in quattro sezioni:

- **Home:** per gli utenti e gli anonimi la home page corrisponde alla ricerca campi, per le strutture invece la pagina dedicata alla gestione dei campi.
- **Profilo:** permette agli utenti e alle strutture di visualizzare e modificare il proprio profilo; l'interfaccia è simile per entrambi gli utenti ma con campi di inserimento diversi.
- **Prenotazioni:** permette agli utenti e alle strutture di gestire le prenotazioni; l'interfaccia è simile per entrambi gli utenti. La differenza risiede nel fatto che gli utenti visualizzano le prenotazioni divise in "future" e "passate" mentre le strutture in aggiunta hanno la possibilità di esportare.
- **Crea Campo:** visibile solo alle strutture, è una pagina che permette di creare un campo.
- **Login/Signup:** visibile soltanto dagli utenti anonimi.
- **Logout:** visibile da tutti gli utenti loggati.

Seguono alcuni *screenshot*.

Parte del risultato

The screenshot shows a user interface for searching sports fields. At the top, there is a navigation bar with links for "Home", "Profilo", "Prenotazioni", and "Logout". Below the navigation bar is a search form titled "Cerca campi". The search form contains four input fields: "Modena" (selected), "Calcio a 5" (selected), "Non specificato" (selected), and "Recensioni". A blue "Cerca" button is located at the bottom right of the search form. The background features a light blue hexagonal pattern.

Pannello Admin

Creata da Alberto Nuzzaci
Matr. 165351
Esame di Tecnologie Web

Figura 3 - Home page per utenti e anonimi.

The screenshot shows an administrator's home page displaying a list of sports fields. At the top, there is a navigation bar with links for "Home", "Profilo", "Prenotazioni", "Crea Campo", and "Logout". Below the navigation bar is a section titled "Campi di Centro Sportivo XYZ" containing four entries:

Nome	Copertura	Prezzo	Caratteristiche	Valutazione	Azione
Calcio a 5	Coperto	45,00€	Spogliatoi, Docce, illuminazione notturna, parcheggio gratuito	4,3 ★	Elimina
Pallavolo	Scoperto	20,00€	Spogliatoi, Docce, parcheggio gratuito, casacche in prestito, tribuna	3,3 ★	Elimina
Tennis	Scoperto	20,00€	Spogliatoi, Docce, illuminazione notturna, parcheggio gratuito, noleggio racchette	3,2 ★	Elimina
Pallavolo	Scoperto	19,00€	Spogliatoi, Docce, parcheggio gratuito, casacche in prestito	2,8 ★	Elimina

Pannello Admin

Creata da Alberto Nuzzaci
Matr. 165351
Esame di Tecnologie Web

Figura 4 - Home page per le strutture

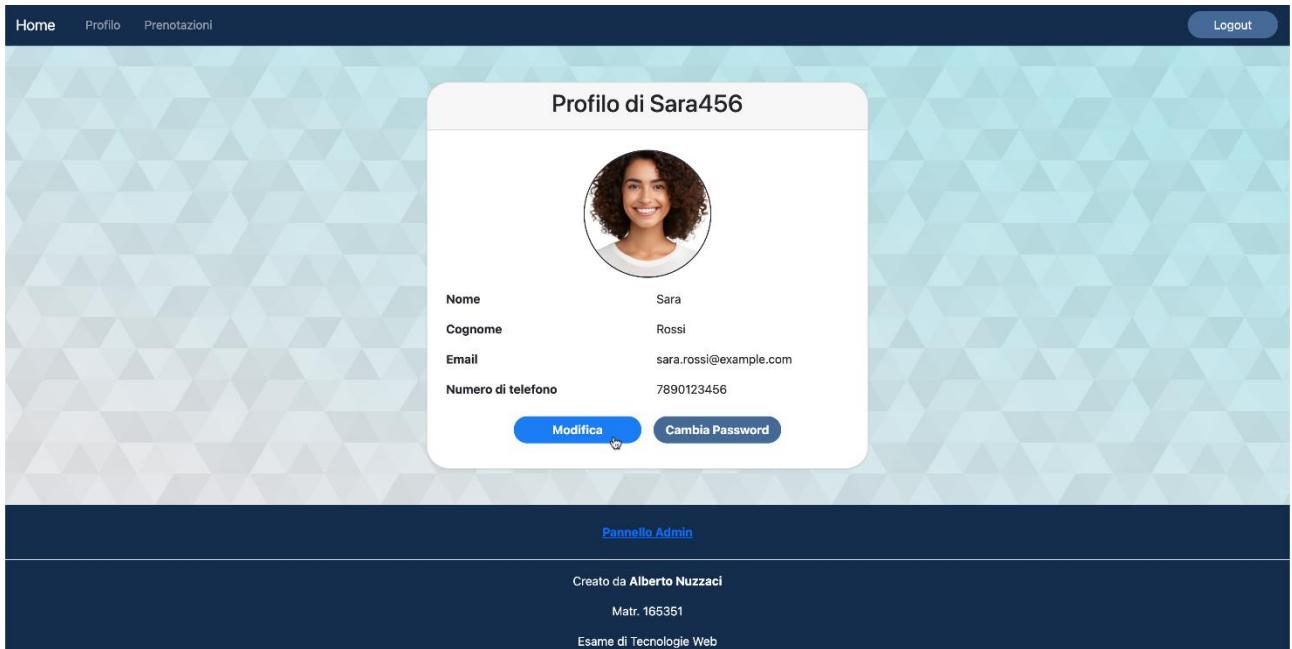


Figura 5 – Sezione profilo degli utenti, le strutture saranno analoghe ma con dati aggiuntivi.

Amministrazione Django

Pagina iniziale > Core > Strutture

AUTENTICAZIONE E AUTORIZZAZIONE	
Gruppi	+ Aggiungi
CORE	
Campi	+ Aggiungi
Prenotazioni	+ Aggiungi
Recensioni	+ Aggiungi
Servizi	+ Aggiungi
Strutture	+ Aggiungi
USERS	
Proprietari Strutture	+ Aggiungi
Users	+ Aggiungi
Utenti	+ Aggiungi

Scegli struttura da modificare

Azioni: ✓ ----- Cancella Strutture selezionate Verifica le strutture selezionate

AZIONE	INDIRIZZO	NUM CIVICO	EMAIL	NUMTELEFONO	VERIFIED
<input type="checkbox"/> NO	Via Torino	120	info@healthfitness.com	+39331123344	✗
<input type="checkbox"/> Health & Fitness	Roma	110	info@fitnessone.it	+390223344556	✗
<input type="checkbox"/> Fitness One	Milano	110	info@wellnesshub.it	+39331112223	✗
<input checked="" type="checkbox"/> Wellness Hub	Modena	100	info@activegym.it	+39331123344	✗
<input type="checkbox"/> Active Gym	Torino	90	info@fitnesszone.com	+393312223344	✗
<input checked="" type="checkbox"/> Fitness Zone	Roma	80	info@sporthealth.com	+393314455667	✗
<input checked="" type="checkbox"/> Sport & Health	Milano	70	info@healthclub.it	+393332224455	✗
<input checked="" type="checkbox"/> Health Club	Modena	60	info@fitnessfactory.com	+390112223344	✗
<input type="checkbox"/> Fitness Factory	Torino	55	info@gymandmore.com	+393336778899	✗
<input type="checkbox"/> Gym & More	Roma	45	info@arenasportiva.it	+390987651234	✗
<input checked="" type="checkbox"/> Arena Sportiva	Napoli	30	info@sportcity.com	+392223344556	✗
<input type="checkbox"/> Sport City	Milano	1	info@wellness.it	+39334455667	✗
<input checked="" type="checkbox"/> Centro Wellness	Roma	22	info@fitclub.it	+390987654321	✗
<input type="checkbox"/> Fit club	Milano	5	info@fitnessplus.com	+39112223334	✗
<input checked="" type="checkbox"/> Palestra Fitness Plus	Napoli	15	info@poli.it	1234567890	✗
<input type="checkbox"/> Polisportiva	Milano	25	info@centrosportivoxyz.com	+391234567890	✗
<input type="checkbox"/> Centro Sportivo XYZ	Roma	10			✗

16 Strutture

VERIFIED

✗
✗
✓
✗
✓
✗
✓
✗
✓
✗
✓
✗
✓
✗
✓
✗
✓
✗
✓
✗

Figura 7 - Pannello admin, notare l'azione di modifica delle strutture

Figura 6 - Strutture verificate

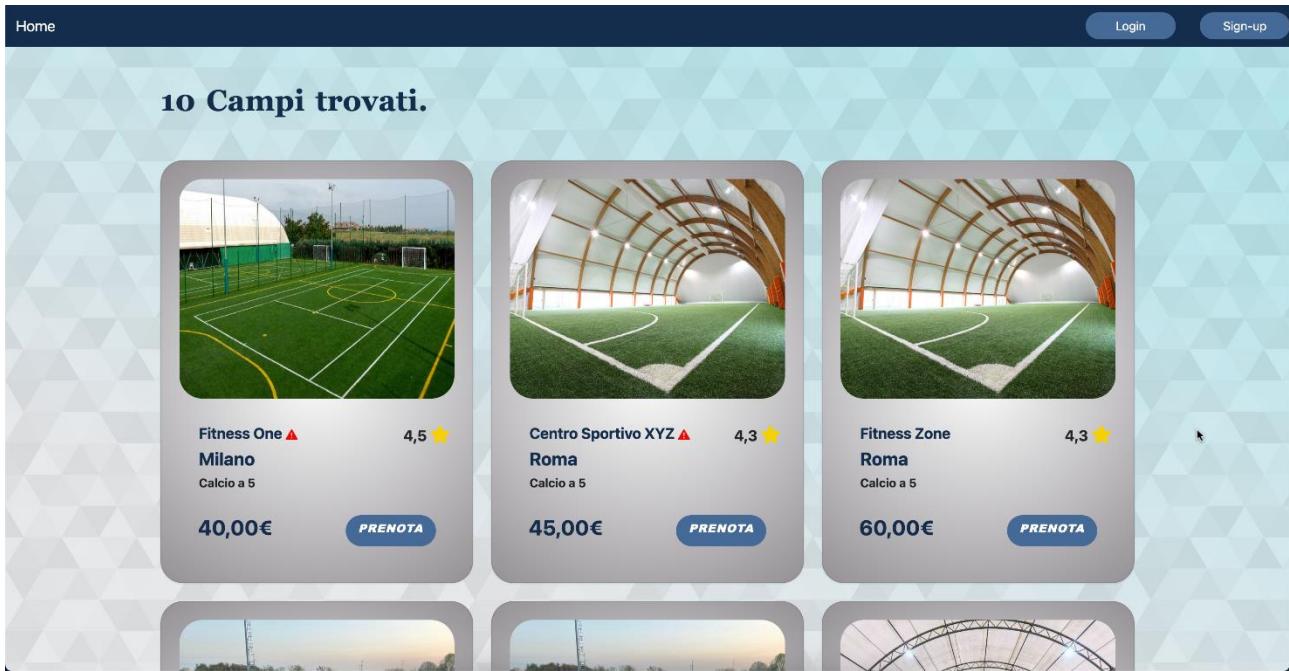


Figura 8 - Lista campi ottenuti dopo una ricerca

Figura 9-10 – pagina dedicata alla prenotazione del campo.

NB: tra i campi consigliati due hanno le immagini uguali ma, come si nota dai nomi, sono campi diversi. Questo è dovuto a una popolazione con immagini fittizie iniziali del database.

Prenotazioni passate					
Data: 02 Luglio 2024	Ora: 10:00	Campo: Pallavolo	Costo: 20,00€	Struttura: Centro Sportivo XYZ	
Data: 02 Luglio 2024	Ora: 10:00	Campo: Pallavolo	Costo: 19,00€	Struttura: Centro Sportivo XYZ	
Data: 02 Luglio 2024	Ora: 10:00	Campo: Pallavolo	Costo: 21,00€	Struttura: Fitness One	
Data: 02 Luglio 2024	Ora: 10:00	Campo: Calcio a 5	Costo: 40,00€	Struttura: Fitness One	
Data: 02 Luglio 2024	Ora: 13:00	Campo: Tennis	Costo: 25,00€	Struttura: Fitness Zone	
1 ... 7 Successiva					
Prenotazioni future					
Data: 06 Luglio 2024	Ora: 10:00	Campo: Tennis	Costo: 20,00€	Struttura: Centro Sportivo XYZ	
Data: 06 Luglio 2024	Ora: 12:00	Campo: Beachsoccer	Costo: 27,00€	Struttura: Fitness One	
Data: 06 Luglio 2024	Ora: 15:00	Campo: Padel	Costo: 32,00€	Struttura: Fitness Zone	
Data: 06 Luglio 2024	Ora: 15:00	Campo: Beach Volley	Costo: 22,00€	Struttura: Polisportiva	
Data: 06 Luglio 2024	Ora: 16:00	Campo: Calcio a 7	Costo: 35,00€	Struttura: Arena Sportiva	
1 ... 6 Successiva					

Figura 11 – prenotazioni lato utente divise in passate e future. Quelle passate che corrispondono ad un campo non ancora recensito hanno il tasto “Recensione” abilitato.

Creazione Campo

Immagine <input type="file" value="Scegli file"/> Nessun file selezionato	Tipo di sport: <input type="text" value="Tennis"/> Copertura: <input type="text" value="Scoperto"/> Costo: <input type="text"/> Servizi Spogliatoi <input type="checkbox"/> Docce <input type="checkbox"/> Illuminazione notturna <input type="checkbox"/> Armadietti <input type="checkbox"/> Bar <input type="checkbox"/> Parcheggio gratuito <input type="checkbox"/> Casacche in prestito <input type="checkbox"/> Noleggio racchette <input type="checkbox"/> Tribuna <input type="checkbox"/> Piscina <input type="checkbox"/>
<input type="button" value="Conferma"/>	

Figura 12 – creazione campo lato Struttura.