SPORT BOOKING: DOCUMENTAZIONE

Sistema di prenotazione campi sportivi con Django

Autore: Tommaso Fachin

Corso e docente: Tecnologie web, docente Nicola Capodieci

Data inizio progetto: 3 maggio 2025

INDICE:

1 Introduzione

1.1 Obiettivi del progetto

Lo scopo di questo progetto è realizzare una piattaforma web per la gestione delle prenotazioni di impianti sportivi, accessibile sia da utenti che da amministratori.
Il progetto è stato sviluppato utilizzando il framework Django, con l'obiettivo di approfondire le competenze su sviluppo web backend.

1.2 Tecnologie usate

Sono state utilizzate le seguenti tecnologie:

- ➔ Python: Come linguaggio di programmazione principale
- ➔ Django: Framework utilizzato per lo sviluppo dell'applicazione, gestione modelli, viste, autenticazione ecc.
- ➔ SQLite: Database predefinito di Django
- ➔ HTML: struttura delle pagine web
- ➔ CSS: Stili personalizzati
- ➔ Bootstrap: framework front-end utilizzato per lo sviluppo di interfacce web responsive e moderne
- ➔ JavaScript: per interazioni dinamiche (conferme, AJAX)
- ➔ Pipenv: utilizzato per gestire le dipendenze e creare l'ambiente virtuale

1.3 Descrizione sintetica dell'applicazione

L'applicazione è utilizzata per prenotare e gestire i campi da tennis, padel, calcetto di più polisportive.
A seconda del tipo di utente che fa il login (societario o utente normale) si aprirà una schermata di home diversa. Gli utenti societari potranno gestire i vari campi e prenotazioni della società di appartenenza. Gli utenti normali invece potranno cercare un campo (secondo diversi filtri) e prenotarlo. Potranno inoltre vedere le informazioni su tutte le società. Nel prossimo punto specifico tutte le funzionalità.

2 Analisi dei requisiti

2.1 Requisiti funzionali

- ➔ Registrazione e autenticazione utenti: gli utenti devono potersi registrare e autenticare.
- ➔ Visualizzazione disponibilità campi: gli utenti non loggati possono vedere le disponibilità dei campi e le informazioni sulle strutture, ma non prenotare.
- ➔ Prenotazione campi: gli utenti registrati possono prenotare un campo scegliendo sport, data, ora e durata. Un campo non può essere prenotato da più di un utente nella stessa ora lo stesso giorno.
- ➔ Gestione prenotazioni: gli utenti possono vedere le prenotazioni passate e future. Inoltre possono annullare una prenotazione solo se mancano più di 4 ore all'inizio.
- ➔ Pagamento: gli utenti possono scegliere il pagamento online o in struttura
- ➔ Gestione corsi e campi (solo se l'utente si è loggato come utente societario): L'utente può aggiungere o eliminare campi della società. L'utente può aggiungere o eliminare un corso. L'utente societario può riservare campi per i corsi, rendendo quegli slot non prenotabili dagli altri utenti. Infine, l'utente societario può annullare prenotazioni di altri utenti
- ➔ Sistema di ricerca: Gli utenti possono cercare un campo applicando filtri per città, sport, data e ora in cui vogliono giocare

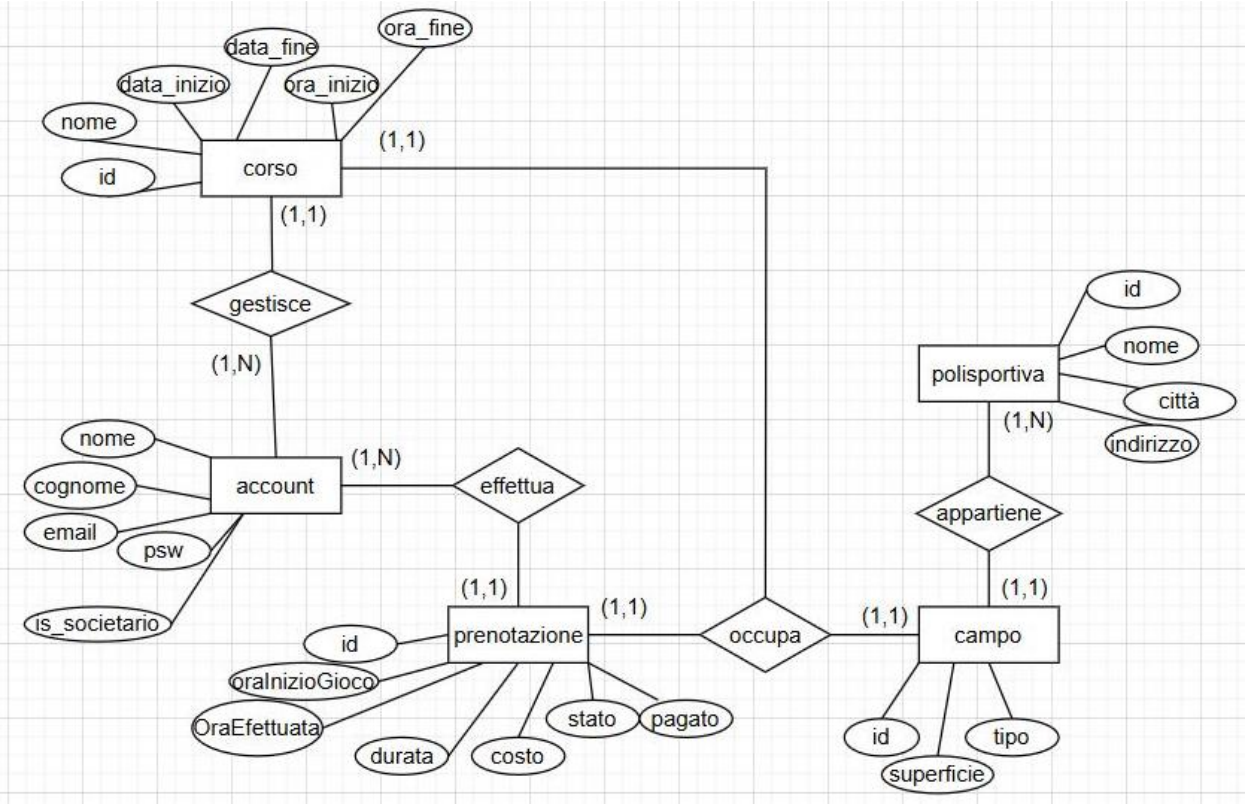
2.2 Requisiti non funzionali

- ➔ Usabilità: L'interfaccia deve essere chiara, semplice e responsive. Utilizzo di Bootstrap per una grafica moderna e uniforme
- ➔ Performance: il sistema deve rispondere rapidamente alle richieste di ricerca e prenotazione
- ➔ Portabilità: Il sistema deve poter essere eseguito su diversi sistemi operativi (Windows, Linux, Mac) grazie all'uso di Python/Django.
- ➔ Utilizzo di git come version control.

3 Progettazione del sistema

3.1 Diagramma E-R

Ecco il diagramma entità-relazione sul quale si basa il database del progetto:



Vincoli chiavi esterne (FK):

FOREIGN KEY (campo.polisportiva_id) REFERENCES (polisportiva.id)

FOREIGN KEY (corso.campo_id) REFERENCES (campo.id)

FOREIGN KEY (corso.account_id) REFERENCES (account.id)

FOREIGN KEY (prenotazione.account_id) REFERENCES (account.id)

FOREIGN KEY (prenotazione.campo_id) REFERENCES (campo.id)

3.2 Modelli (models.py)

Descrizione delle entità (classi):

- Account: rappresenta un utente del sistema, con attributi come nome, cognome, email, password, ruolo (is_societario), e associazione a una polisportiva.
- Polisportiva: rappresenta una società sportiva, con nome, città e indirizzo.
- Campo: rappresenta un campo sportivo, con nome, tipo (tennis, padel, calcio a 5), superficie e associazione a una polisportiva.
- Corso: rappresenta un corso organizzato dalla polisportiva, con nome, campo, date e orari di inizio/fine, e account responsabile.
- Prenotazione: rappresenta una prenotazione di un campo, con data, ora, durata, stato, costo, pagato, e associazione a un account e a un campo.

3.3 Struttura del progetto django

App create:

- Accounts: Gestisce utenti, autenticazione, profili, ruoli (utente normale e societario), e la logica di gestione corsi e campi
- Prenotazioni: gestisce la creazione, visualizzazione e cancellazione dei campi (home)

3.4 Template principali

Ecco un elenco dei template principali del progetto:

- ✓ home.html: pagina di ricerca e accesso rapido alle funzionalità principali.
- ✓ register.html, accounts.html: registrazione e login.
- ✓ campi_polisportiva.html: visualizzazione e prenotazione dei campi.
- ✓ mie_prenotazioni.html: gestione delle prenotazioni dell'utente.
- ✓ gestione_corsi.html, gestione_campi.html, gestione_polisportiva_home: gestione riservata agli utenti societari.

3.5 Tecnologie Front-end

Ecco l'elenco delle tecnologie utilizzate per il Front-End:

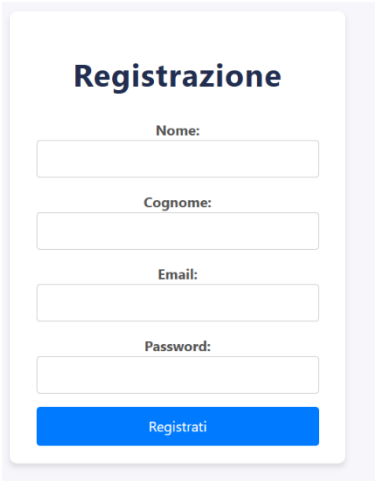
- ✓ HTML5: struttura delle pagine.
- ✓ CSS3: stili personalizzati.
- ✓ Bootstrap 5: framework per layout responsive.
- ✓ Bootstrap Icons: icone vettoriali.
- ✓ JavaScript: interazioni dinamiche (es. conferme, AJAX).
- ✓ Django Templates: generazione dinamica delle pagine HTML.

4 Implementazione

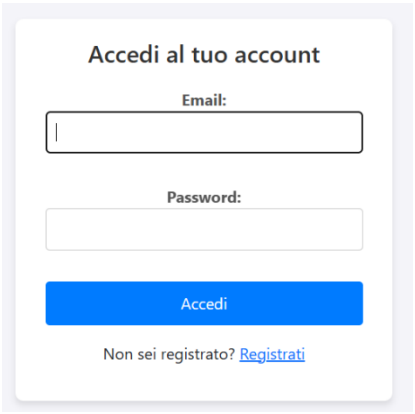
4.1 Spiegazione funzionalità principali

Ecco la home del sito

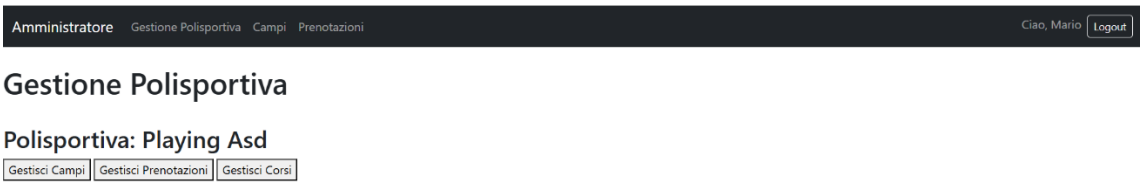
Nella pagina di registrazione avremo questo form di registrazione:



Se invece dalla home si preme su login comparirà un form per loggarsi:



A seconda che l'utente loggato sia un utente societario o un utente normale potrà fare cose diverse.
Vista utente societario:

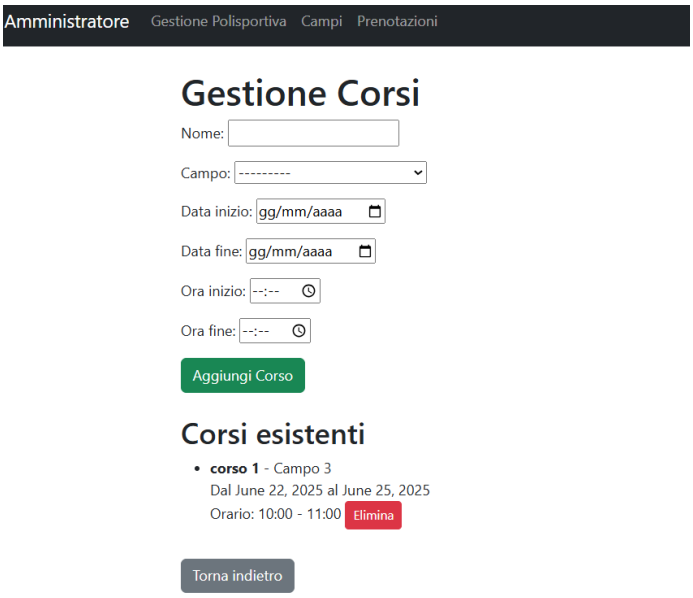


A seconda di cosa sceglie l'utente societario gli si aprono 3 schede diverse

Pagine di gestione dei campi:

qui potrà inserire un nuovo campo o eliminare uno dei campi già esistenti della polisportiva di appartenenza

Qui si possono annullare le prenotazioni ad altri utenti

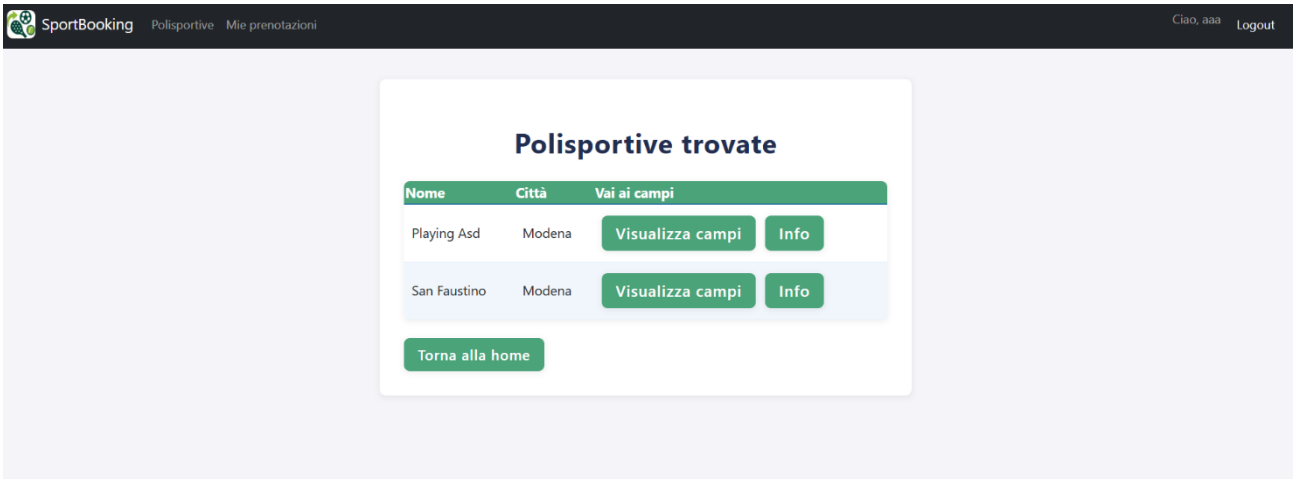


Nella pagina di gestione corsi invece si possono aggiungere o eliminare corsi

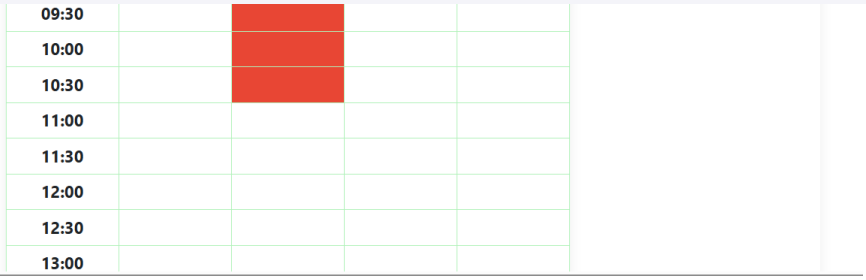
Dopo aver effettuato il logout ed aver fatto accesso con l'utente normale verrà aperta la pagina con la home iniziale

Qui ci sono i filtri di selezione dei campi, ma funziona anche se l'utente non inserisce nulla e preme su cerca. In questo caso farà vedere tutte le polisportive.

Se si preme sul bottone visualizza campi verranno fatti vedere i campi con le rispettive disponibilità



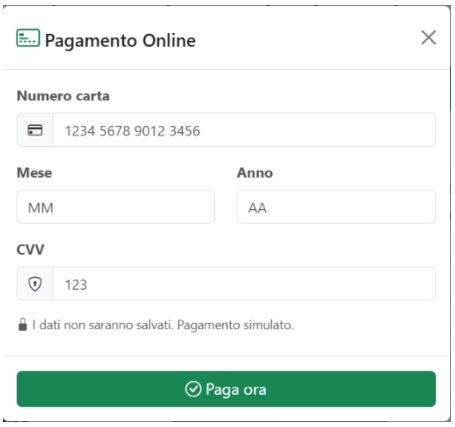
Gli slot



selezionati in rosso vuol dire che sono già occupati. È possibile selezionare gli slot che si vogliono prenotare e poi selezionare la modalità di pagamento con il menu a tendina a destra dei campi. Se si seleziona online allora comparirà un form per inserire gli estremi della carta di credito. Al termine del pagamento verra mandata una mail di conferma

Inoltre, è possibile cambiare giorno, e all'occorrenza se si preme sul calendario si potrà selezionare il campo dal calendario

Form per pagamento online



Infine, se si va su Mie prenotazioni, si possono visualizzare tutte le prenotazione effettuate da questo account (sia passate che future). Per le prenotazioni future è possibile disdirle se mancano più di 4 ore all'inizio.

5 Gestione e memorizzazione dati:

5.1 Struttura file models.py

Nel file models.py abbiamo le varie classi che rappresentano le tabelle del database. Nello specifico :

- ➔ La classe AccountManager serve a Django per sapere come creare utenti e superuser. Non è presente nel modello del database.
- ➔ La classe Account
- ➔ La classe Polisportiva
- ➔ La classe Campo
- ➔ La classe Corso
- ➔ La classe Prenotazione

5.2 Popolazione Database

Per prima cosa se si vuole essere inseriti come utenti societari bisogna fare richiesta all'amministratore del sito che aggiungerà l'utente manualmente. Stessa cosa se si vuole inserire una nuova società.

Per motivi di test ho già popolato il database con i seguenti dati (lascio qui sotto le foto del file .json utilizzato per importare i dati nel database):

Polisportive: Utenti societari:

```
[
  {
    "model": "accounts.polisportiva",
    "pk": 1,
    "fields": {
      "nome": "PlayingASD",
      "città": "Modena",
      "indirizzo": "Via Vittorio Padovani, 45",
      "telefono": "3339970364",
      "email": "info@playinasd.it"
    }
  },
  {
    "model": "accounts.polisportiva",
    "pk": 2,
    "fields": {
      "nome": "San Faustino",
      "città": "Modena",
      "indirizzo": "Via Wiligelmo, 72",
      "telefono": "3339970364",
      "email": "info@sanfaustino.it"
    }
  },
  {
    "model": "accounts.polisportiva",
    "pk": 3,
    "fields": {
      "nome": "Sacca",
      "città": "Modena",
      "indirizzo": "Via Alfonso Paltrinieri, 80",
      "telefono": "3339970364",
      "email": "info@sacca.it"
    }
  },
  {
    "model": "accounts.polisportiva",
    "pk": 4,
    "fields": {
      "nome": "Sporting Sassuolo",
      "città": "Sassuolo",
      "indirizzo": "Via Vandelli, 25",
      "telefono": "3339970364",
      "email": "info@sportingsassuolo.it"
    }
  },
  {
    "model": "accounts.polisportiva",
    "pk": 5,
    "fields": {
      "nome": "Sporting Formigine",
      "città": "Formigine",
      "indirizzo": "Via Caduti di Superga, 2",
      "telefono": "3339970364",
      "email": "info@sportingformigine.it"
    }
  },
```

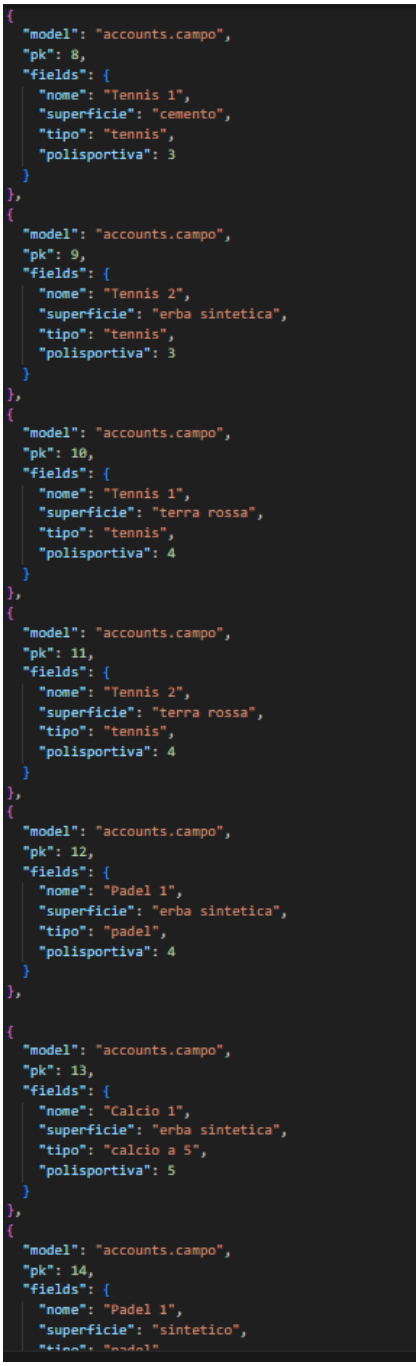
```
{
  "model": "accounts.account",
  "pk": 1,
  "fields": {
    "nome": "Mario",
    "cognome": "Rossi",
    "email": "societario@playingasd.it",
    "password": "pbkdf2_sha256$260000$test$testhash",
    "is_societario": true,
    "polisportiva": 1,
    "is_active": true,
    "is_staff": false
  }
},
{
  "model": "accounts.account",
  "pk": 2,
  "fields": {
    "nome": "Luca",
    "cognome": "Bianchi",
    "email": "societario@sanfaustino.it",
    "password": "pbkdf2_sha256$260000$test$testhash",
    "is_societario": true,
    "polisportiva": 2,
    "is_active": true,
    "is_staff": false
  }
},
{
  "model": "accounts.account",
  "pk": 3,
  "fields": {
    "nome": "Giulia",
    "cognome": "Verdi",
    "email": "societario@sacca.it",
    "password": "pbkdf2_sha256$260000$test$testhash",
    "is_societario": true,
    "polisportiva": 3,
    "is_active": true,
    "is_staff": false
  }
},
{
  "model": "accounts.account",
  "pk": 4,
  "fields": {
    "nome": "Anna",
    "cognome": "Neri",
    "email": "societario@sportingsassuolo.it",
    "password": "pbkdf2_sha256$260000$test$testhash",
    "is_societario": true,
    "polisportiva": 4,
    "is_active": true,
    "is_staff": false
  }
},
{
  "model": "accounts.account",
  "pk": 5,
  "fields": {
    "nome": "Paolo",
    "cognome": "Blu",
    "email": "societario@sportingformigine.it",
    "password": "pbkdf2_sha256$260000$test$testhash",
    "is_societario": true,
    "polisportiva": 5,
    "is_active": true,
```

Elenco campi 1:

```
{
  "model": "accounts.campo",
  "pk": 1,
  "fields": {
    "nome": "Calcetto 1",
    "superficie": "erba sintetica",
    "tipo": "Calcio a 5",
    "polisportiva": 1
  }
},
{
  "model": "accounts.campo",
  "pk": 2,
  "fields": {
    "nome": "Tennis 1",
    "superficie": "terra rossa",
    "tipo": "tennis",
    "polisportiva": 1
  }
},
 {
  "model": "accounts.campo",
  "pk": 3,
  "fields": {
    "nome": "Tennis 2",
    "superficie": "erba sintetica",
    "tipo": "tennis",
    "polisportiva": 1
  }
},
 {
  "model": "accounts.campo",
  "pk": 4,
  "fields": {
    "nome": "Tennis 3",
    "superficie": "erba sintetica",
    "tipo": "tennis",
    "polisportiva": 1
  }
},

  "model": "accounts.campo",
  "pk": 5,
  "fields": {
    "nome": "Padel 1",
    "superficie": "erba sintetica",
    "tipo": "padel",
    "polisportiva": 1
  }
},
{
  "model": "accounts.campo",
  "pk": 6,
  "fields": {
    "nome": "Calcio 1",
    "superficie": "erba",
    "tipo": "Calcio a 5",
    "polisportiva": 2
  }
},
{
  "model": "accounts.campo",
  "pk": 7,
  "fields": {
    "nome": "Calcio 2",
    "superficie": "sintetico",
    "tipo": "Calcio a 5",
```

Elenco campi 2:



```
{
  "model": "accounts.campo",
  "pk": 8,
  "fields": {
    "nome": "Tennis 1",
    "superficie": "cemento",
    "tipo": "tennis",
    "polisportiva": 3
  }
},
{
  "model": "accounts.campo",
  "pk": 9,
  "fields": {
    "nome": "Tennis 2",
    "superficie": "erba sintetica",
    "tipo": "tennis",
    "polisportiva": 3
  }
},
{
  "model": "accounts.campo",
  "pk": 10,
  "fields": {
    "nome": "Tennis 1",
    "superficie": "terra rossa",
    "tipo": "tennis",
    "polisportiva": 4
  }
},
{
  "model": "accounts.campo",
  "pk": 11,
  "fields": {
    "nome": "Tennis 2",
    "superficie": "terra rossa",
    "tipo": "tennis",
    "polisportiva": 4
  }
},
{
  "model": "accounts.campo",
  "pk": 12,
  "fields": {
    "nome": "Padel 1",
    "superficie": "erba sintetica",
    "tipo": "padel",
    "polisportiva": 4
  }
},

{
  "model": "accounts.campo",
  "pk": 13,
  "fields": {
    "nome": "Calcio 1",
    "superficie": "erba sintetica",
    "tipo": "calcio a 5",
    "polisportiva": 5
  }
},
{
  "model": "accounts.campo",
  "pk": 14,
  "fields": {
    "nome": "Padel 1",
    "superficie": "sintetico",
```

5.3 Comandi utili per popolamento ed eliminazione dati nel database

Per eliminare velocemente i dati presenti nel database utilizzare il comando:

- ➔ python manage.py flush

Per caricare invece i dati del file .json utilizzare il comando:

- ➔ python manage.py loaddata dati_iniziali.json

** il database si può ripulire anche eliminando il file db.sqlite3. in tal caso bisognerà poi rifare le migrazioni

6. Test

6.1 Spiegazione test scelti

Ho scelto di fare 2 test nel mio progetto:

1) il primo testa verifica che il codice per la creazione di una prenotazione funzioni correttamente: controlla che i dati siano salvati giusti e che la prenotazione risulti pagata

2) il secondo test controlla che la vista campi_polisportiva funzioni: verifica che la pagina risponda correttamente (HTTP 200) e che mostri il nome della polisportiva, cioè che la pagina venga generata e visualizzata come previsto

6.2 Implementazione

Ecco l'implementazione del primo test:

```python
#test la creazione di una prenotazione
class PrenotazioneModelTest(TestCase):
    def setUp(self):
        self.polisportiva = Polisportiva.objects.create(  #crea una polisportiva di test
            nome="Test ASD", città="Modena", indirizzo="Via Test", telefono="1234567890", email="test@asd.it"
        )
        self.account = Account.objects.create(   # crea un account di test
            nome="Mario", cognome="Rossi", email="mario@rossi.it", password="test", is_societario=False, is_active=True
        )
        self.campo = Campo.objects.create(  #campo di test
            nome="Campo 1", superficie="erba", tipo="calcio", polisportiva=self.polisportiva
        )

    def test_creazione_prenotazione(self):   #definizione del vero e proprio test
        pren = Prenotazione.objects.create(
            campo=self.campo,
            account=self.account,
            data=date.today(),
            ora_inizio=time(10, 0),
            ora_fine=time(11, 0),
            durata=60,
            costo=20.0,
            stato='accettata',
            pagato=True
        )
        self.assertEqual(pren.campo.nome, "Campo 1")
        self.assertEqual(pren.account.email, "mario@rossi.it")
        self.assertTrue(pren.pagato)
```

Ecco l'implementazione del secondo test:

```python
#test la vista: campi_polsportiva
class CampiPolisportivaViewTest(TestCase):
    def setUp(self):
        self.polisportiva = Polisportiva.objects.create(
            nome="Test ASD", città="Modena", indirizzo="Via Test", telefono="1234567890", email="test@asd.it"
        )

    def test_campi_polisportiva_view(self):
        url = reverse('campi_polisportiva', args=[self.polisportiva.id])
        response = self.client.get(url)
        self.assertEqual(response.status_code, 200)
        self.assertContains(response, "Test ASD")
```

6.3 Risultati

Se, dopo aver eseguito i test con il comando: python manage.py test accounts, python non segnala errori (Errors o Failed) vuol dire che i test sono andati a buon fine, come in questo caso.

```
(Prenotazioni-Qy49XoZe) (Prenotazioni) D:\tommaso\UNIVERSITA\Esami\Progetto-sito-Prenotazioni
\Prenotazioni>python manage.py test accounts
Found 2 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..
----------------------------------------------------------------------
Ran 2 tests in 0.501s

OK
Destroying test database for alias 'default'...
```