
Chiara Boni - Matr. 97120

Progetto “Linguaggi Dinamici”

# TRACCIA PROGETTO

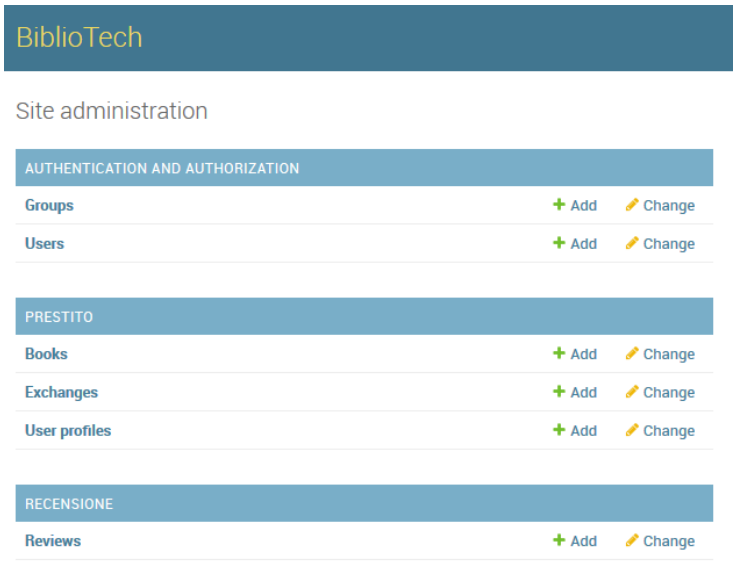
**BiblioTech** è il progetto che realizza una **biblioteca virtuale** che simula lo scambio di libri in formato Ebook.
È possibile scorrere il catalogo dei libri disponibili, leggere le recensioni degli utenti che hanno letto in precedenza il libro selezionato e, ovviamente, prendere in prestito il libro stesso.

Si è strutturata l'intera biblioteca su **vari** macro-ambienti:

- Utente generico
- Utente registrato
- Utente amministratore


Un **utente generico** ha l'accesso più limitato al sito, in quanto può solamente accedere ai singoli dati dei libri e leggere le recensioni dei lettori prima di lui.
Chiaramente può creare un account e registrarsi al sito.
Se è già in possesso delle credenziali di accesso allora gli viene data la possibilità di effettuare il **login**.


Un **utente registrato** è in grado di consultare i libri e prenderli in **prestito** (di massimo 30 giorni).
Non esiste un limite sul numero di libri che può leggere in contemporanea.
Inoltre, questo tipo di utente possiede un proprio **profilo** nel quale modificare le proprie credenziali quali nome, cognome, username e password.
Nel profilo del singolo utente è permesso anche visionare i libri all'attivo che quest'ultimo deve ancora riportare.
Al momento della **restituzione** verrà richiesto all'utente se vuole rilasciare una **recensione** o meno.
Il sistema non accetta ritardi nella consegna del libro: se l'utente non ha consegnato il libro entro la scadenza, questo verrà eliminato in maniera automatica dal profilo stesso.




L'**utente amministratore** ha il completo controllo sull'intero progetto.

In primo luogo può, ovviamente, visionare i dati di tutto il progetto. È in grado di inserire, modificare oppure eliminare ogni istanza di ogni entità.

Per poter avere questo tipo di privilegi bisogna avere un profilo di tipo 'super user' e, quindi, avere i permessi per modificare il DB che gestisce i dati del progetto.

---

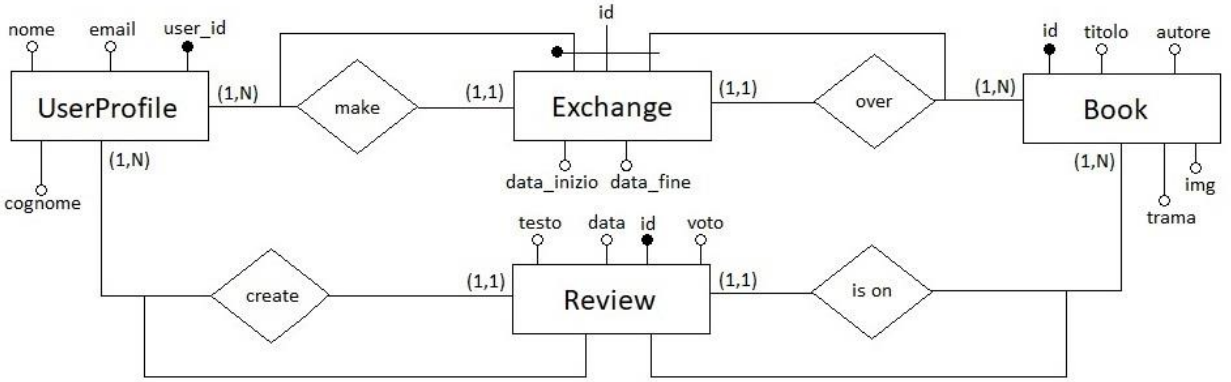
# ORGANIZZAZIONE CODICE

## 1. ORGANIZZAZIONE DATABASE

Per sviluppare questo progetto ho utilizzato il database di **PostgreSQL**, nel quale ho creato un utente 'adminbiblio' al quale potessi garantire tutti i privilegi di accesso e modifica ai dati.

Come ambiente di sviluppo Python-Django ho utilizzato **PyCharm**.
Per poter garantire l'accesso alla pagina 'admin' dedicata, occorre modificare il file **settings.py** e creare il collegamento con la base di dati appena creata.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'bibliotech',
        'USER': 'adminbiblio',
        'PASSWORD': 'adminpassword',
        'HOST': '',
        'PORT': ''
    }
}
```




In secondo luogo si modella la traccia del progetto mediante un **modello E-R**.
Un **utente** registrato possiede il proprio profilo identificativo, reso univoco dalla presenza di un id numerico progressivo.
Presenta dati anagrafici quali nome, cognome ed un indirizzo di posta elettronica.
Anche i **libri** sono identificati da un numero progressivo, in aggiunta dei dati che possono interessare i futuri lettori, quali titolo, autore, trama e immagine di copertina.
Ogni utente è in grado di effettuare uno o più **prestiti** di libri.
Rispettivamente, il singolo prestito è vincolato da un id numerico univoco progressivo e dai dati identificativi dell'utente e del libro.
Non esistono vincoli sul numero di libri che può avere all'attivo, in contemporanea, un utente.
Più utenti possono prendere in prestito lo stesso libro in quanto, trattandosi di Ebook, non esiste un limite fisico di copie disponibili.
Un singolo utente può riprendere in prestito lo stesso libro, ma solo dopo averlo riportato e quindi cessato il prestito precedente.
Il sistema non accetta ritardi in fase di consegna: se l'utente non riporta il libro entro la scadenza, il sistema lo eliminerà dal suo profilo in maniera automatica.
Una volta riportato un libro, il sistema darà la possibilità all'utente di rilasciare una **recensione**.
Queste sono identificate da un numero univoco, data, testo e voto compreso da 1 e 5 indicate il grado di apprezzamento dell'utente stesso.
Un utente può scrivere più recensioni sullo stesso libro, se lo prende in prestito più volte.
Le recensioni sono pubbliche: possono essere viste da ogni utente (registrato e non) in modo tale da invogliare o meno lettori futuri.

Una volta tradotta in Python, ogni entità diventa una **classe**.

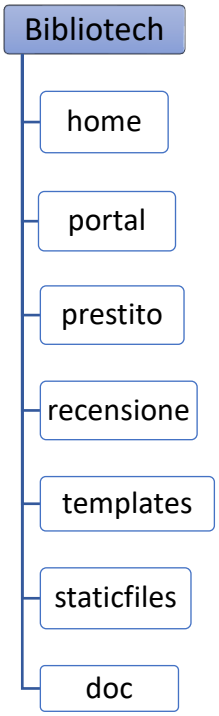
| UserProfile |
| --- |
| - user_id: int |
| - email: str |
| - nome: str |
| - cognome: str |

| Exchange |
| --- |
| - user_id: int |
| - book_id: int |
| - data_inizio: date |
| - data_fine: date |

| Book |
| --- |
| - id: int |
| - titolo: str |
| - trama: str |
| - autore: str |
| - img: str |

| Review |
| --- |
| - user_id: int |
| - book_id: int |
| - data: date |
| - testo: str |
| - voto: int |

## 2. DIVISIONE IN CARTELLE



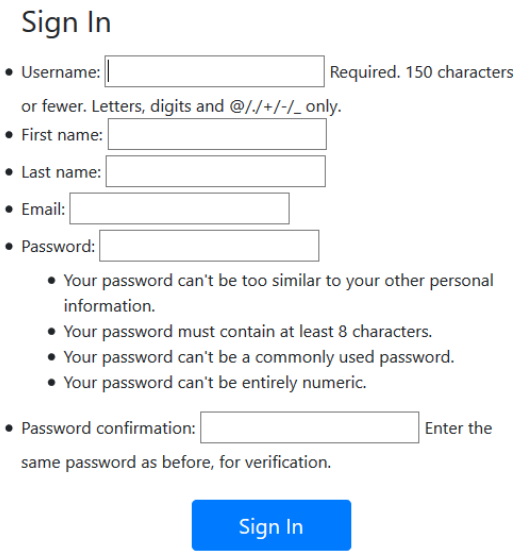

Per avere una divisione delle operazioni più chiare, si è diviso il progetto in più **applicazioni** tali che gestiscano singolarmente vari aspetti.

- **Home**: gestisce la home page e l'indirizzamento iniziale;
- **Portal**: questa applicazione presenta le view per gestire la registrazione degli utenti e la amministrazione del profilo, comprese le operazioni per la modifica delle credenziali.
- **Prestito**: gestisce la creazione del prestito e la relativa restituzione. È presente inoltre il metodo per la restituzione automatica in base alla scadenza.
- **Recensione**: si incarica di creare e gestire le recensioni.
- **Templates**: presenta i file html relativi alle singole applicazioni.
- **Staticfiles**: raccolta di tutti i file statici, quali immagini, file CSS e file Javascript.
- **Doc**: presenta gli elementi di documentazione.

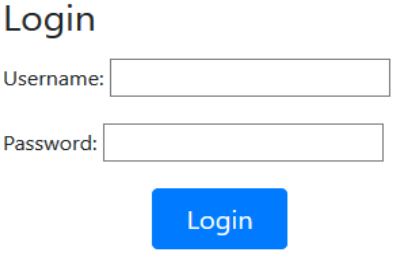
## 3. LOGICA REGISTRAZIONE / LOGIN

Un ruolo fondamentale nel progetto è rappresentato dalle logiche di **registrazione** e di **login**.

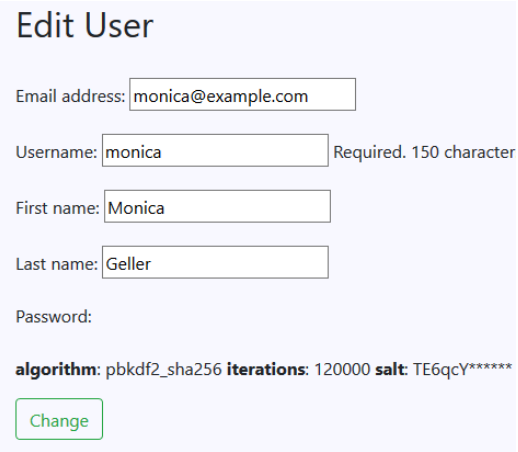
Nel primo caso si è costruita una classe **RegistrationForm** che eredita dalla classe di base di Django "UserCreationForm"; è stata ampliata in modo da incorporare anche il campo 'email' e, dunque, essere inserito nel database insieme alle altre credenziali dell'utente.

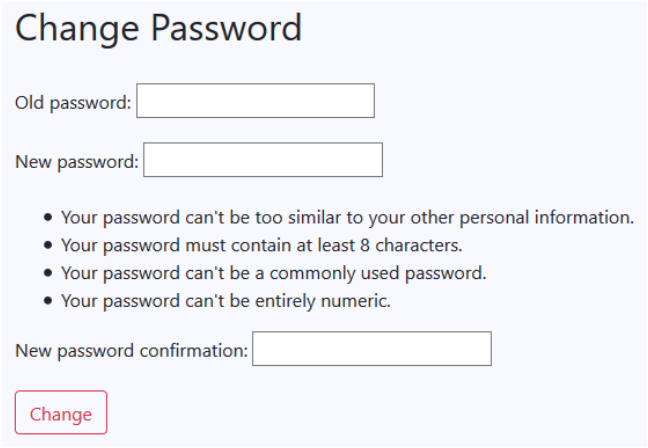



La logica di **login** presenta un form molto semplice, infatti viene utilizzato il 'AuthenticationForm' che Django rende disponibile.
Prende in ingresso l'username e la password dell'utente e, attraverso la funzione **login (request, user)** permette di loggare l'utente all'interno del sito, se i suoi dati sono stati reputati corretti.



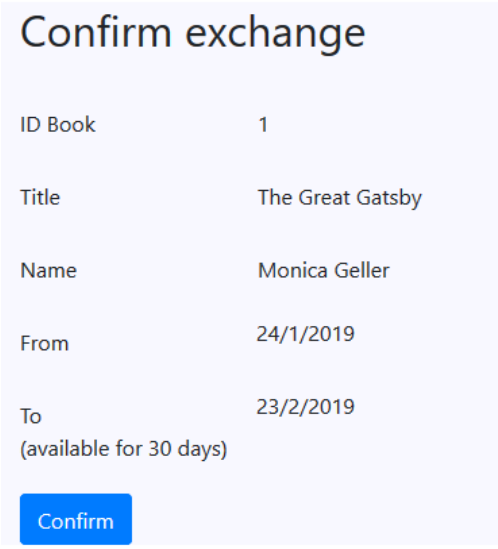
L'aspetto più importante di un **utente registrato** è quello di avere un **profilo** privato nel quale modificare le proprie credenziali e vedere la lista di libri che ha all'attivo.




Nel primo caso, si è creata una classe **EditProfileForm** che eredita i suoi componenti dalla classe Django 'UserChangeForm'.
Ho scelto di nascondere la password, in questo form, perché è un dato estremamente sensibile e, dunque, ho deciso di trattarlo separatamente.
Per modificare la password è stato sufficiente gestire il '**PasswordChangeForm**' che Django mette a disposizione.
Per chiedere un'ulteriore conferma, una volta compilati i campi per modificare la password, verrà mostrato all'utente un pop-up realizzato in **Javascript** che permetterà di annullare l'operazione.

4. **LOGICA PRESTITO E RESTITUZIONE LIBRI**

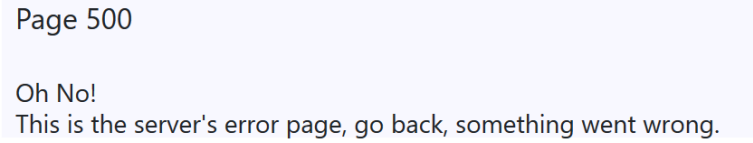



Il primo vincolo fondamentale per il **prestito** all'interno di BiblioTech è che l'utente possegga un account e che abbia già effettuato il login.
L'utente, a questo punto, può navigare all'interno della biblioteca e scegliere quale libro aggiungere al suo profilo personale.

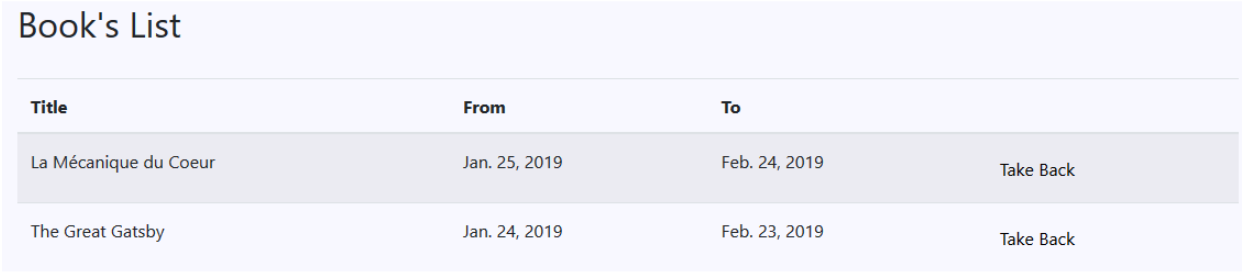
Scelto il volume da aggiungere, apparirà la schermata di **conferma** che riassume i vincoli del prestito, tra cui la scadenza a 30 giorni dal giorno stesso in cui si prende il libro, calcolata dinamicamente.

Più a basso livello, una volta confermati i dati del prestito, verrà creata una nuova istanza della classe '**exchange**', aggiornando il database con i nuovi dati appena creati.
Verrà, inoltre, aggiornato il profilo con l'elenco dei libri che l'utente ha all'attivo.

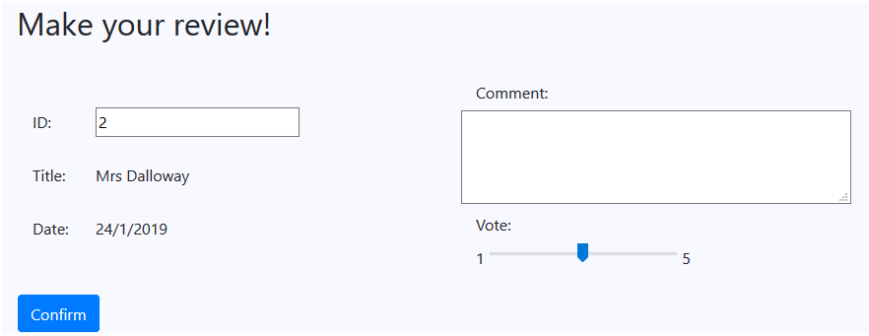
Non esiste un numero massimo di libri da prendere in prestito e lo stesso libro può essere preso dallo stesso utente più volte, ma solo dopo averlo riportato, quindi cessato il prestito tutt'ora in corso.




Se, per errore o distrazione, l'utente prova a ricreare un prestito su un libro che ha attualmente all'attivo, il sito genererà un errore server e dunque la pagina corrispettiva, perché questo tipo di azione va a violare un **vincolo** all'interno della base di dati creata.
Più genericamente, si è sviluppato anche una pagina **400.html** che verrà utilizzata e visualizzata ogni volta che un utente proverà ad accedere ad un indirizzo URL non valido per il sito.




La **restituzione** di un libro viene eseguita dal profilo utente e viene simulata andando ad eliminare il libro selezionato dalla lista dello specifico utente stesso.
Questa lista contiene anche i termini di scadenza del prestito stesso.
A basso livello questo implica l'eliminazione della riga specifica all'interno della tabella 'exchange'.
Il sistema non accetta ritardi: se il libro non viene riportato entro la scadenza, verrà eliminato in maniera **automatica** dalla lista dell'utente.

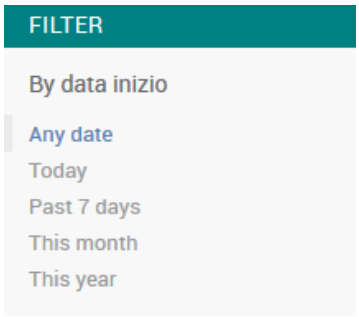



Il sistema dà la possibilità, inoltre, di lasciare **recensioni** sui libri letti; una volta eseguita la restituzione di questo, verrà chiesto all'utente se gradisce lasciare un commento ed una votazione.
Un utente può lasciare più recensioni sullo stesso libro.
Ogni recensione è pubblica e leggibile da un qualsiasi tipo di utente che acceda alla sezione singola del libro selezionato.
Per questo compito ho ideato un form specifico che possa essere salvato, una volta confermato, ed è stato collegato ad una **view** che prende i dati in ingresso e crea una nuova istanza della classe 'review', aggiornando la base di dati coi nuovi dati appena inseriti.

## 5. GESTIONE SEZIONE ADMIN

Questa sezione è accessibile solamente dal **super utente**, precedentemente creato, che ha completi privilegi sulla base di dati.
Django riserva una sezione dedicata per la gestione dell'admin all'indirizzo 127.0.0.1:8000/admin, accessibile appunto mediante le sue credenziali.

Ogni modello di ogni applicazione deve essere aggiunto mediante il comando 'admin.site.register' nel file **admin.py** per essere visibile dal sito principale dell'amministratore.




In questo file ho, inoltre, aggiunto classi che permettessero di specificare i campi di ogni entità da visualizzare, mediante la creazione della lista '**list_display**'.
Per le tabelle di 'exchange' e 'review' ho inserito delle opzioni di '**list_filter**' in base alla data, che possono tornare utili all'amministratore per analizzare l'andamento della sua biblioteca.

Per facilitare l'accesso al singolo libro ho aggiunto un campo di ricerca, '**search_fields**', che ricerca il libro in base al titolo.

Infine, creando il file '**base_site.html**', che eredita l'impaginazione generale da 'admin/base.html', è stato possibile cambiare alcuni aspetti e dare una personalizzazione alla pagina web.

## 6. MIDDLEWARE

Il sistema di **middleware** in Django è un insieme di agganci che vengono resi disponibili tra quelle che sono le richieste e le risposte di oggetti.
Esistono tantissime funzioni che possono essere eseguite da questo software "di mezzo", come per esempio la gestione delle richieste di autenticazione e delle sessioni utente.
Ogni funzione attiva viene specificata nel file **settings.py** dell'applicazione, al campo "middleware".

Ho inserito una specifica funzionalità al progetto: la classe **LoginRequiredMiddleware.**
Permette di gestire un insieme di re indirizzamenti: ogni volta che un utente non loggato tenda di accedere a parti del progetto a lui non disponibili, si viene immediatamente re indirizzati alla pagina di **login**, in modo tale da avere un login forzato e non generare errori nella navigazione.

```
LOGIN_URL = '/portal/'
```
Inserito nel file settings.py, specifica l'indirizzo del form e view di login.

## 7. **TESTING**

La parte di Unit Testing prevede l'inserimento di test automatizzati creati allo scopo di verificare determinati aspetti, reputati critici, all'interno del progetto.
Come scelta implementativa, si è scelto di inserire test che vanno a mirare principalmente sulla coerenza dei dati all'interno del database.

In primo luogo si dividono i test in base all'applicazione specifica e vengono poi inseriti nel file **tests.py**.

**ProfileTest**: classe di test sulla parte di progetto dedicata agli utenti.
In questo test ho deciso di dare la priorità al sistema di **views** dedicate o meno a tipi di utenti diversi.

- `test_valid_login_view`: creato un User generico, questo test verifica se è possibile passare le sue credenziali alla view di login e effettuare l'operazione di accesso.
- `test_view_generic_user_profile`: testa il corretto comportamento della redirezione di un utente non loggato. In questo caso si tenta di accedere ad un indirizzo negato per gli utenti generici.
- `test_view_login_profile`: test speculare al precedente, in questo caso un utente effettua il login e si conferma che possa accedere alla view del suo profilo.

**ExchangeTest**: classe di test dedicati alla componente dei prestiti.
Per questo tipo di verifiche mi sono concentrata principalmente sulla coerenza dei **dati interni** al database.

- `test_all_exchange_dataf_maggiore_datai`: verifica sulla coerenza delle date. Ogni prestito, infatti, deve avere data iniziale minore di quella finale.
- `test_scadenza_maggiore_di_oggi`: si testa l'asserzione inserita in fase di creazione del modello. Non sono ammessi prestiti già scaduti, quindi con data finale passata.

**BookTest:** classe per verificare la sezione del progetto dedicata ai libri.
Per questo test mi sono concentrata sul **modello**.

- `test_create_book`: permette di verificare la correttezza della formulazione del modello. Inserendo dati di esempio, si tenta di creare un oggetto "Book" all'interno del database.

**ReviewTest**: è la classe in cui inserire i test agenti sulla parte delle recensioni.
Anche in questo caso ho preferito concentrarmi sulla coerenza delle informazioni che si sono inserite nel database e, quindi, nel **modello**.

- `test_recensioni_data_passata`: verifica sulle date delle recensioni inserite. Si testa che ogni data sia coerente e quindi verificatesi nel passato.
- `test_voti_negativi_o_maggiori_5`: test che si concentra sul campo dei voti delle recensioni, ovvero verifica che esso rispetti l'intervallo di valori pensato dall'amministratore.

## 8. **SOFTWARE AGGIUNTO**

Ho inserito nel progetto porzioni di software aggiuntivo, per migliorare funzionalità e grafica dell'intero progetto stesso.

- **Psycopg2**: pacchetto aggiuntivo a Django, essenziale per lavorare sul database costruito in PostgreSQL.
- **Pillow**: è l'estensione della libreria PIL, ovvero Python Imaging Library.
  Una immagine può infatti consistere in uno o più insiemi di dati e questa libreria ha lo scopo di immagazzinare queste informazioni in una sola immagine.
  È necessario installare questa libreria per poter gestire i campi ImageField all'interno dei modelli.
- **Bootstrap:** è stato inserito non nella sua forma stand-alone ma tramite il collegamento con il suo foglio di stile CSS, nella pagina 'base.html', in modo tale che tutte le altre pagine html del progetto possano ereditarlo.
  Bootstrap è un componente di front-end grafico che permette di modellare pagine web, inserendo form predefiniti e controlli ulteriori.
- **WhiteNoise:** è un servizio Python che permette di gestire i propri file statici – immagini, file CSS, file Javascript – per le applicazioni web.
- **SweetAlert:** è un componente di front-end grafico che permette di incorporare codice Javascript, per le applicazioni, tale che possa essere più sicuro.