Django

Documentazione

Scrivere la tua prima applicazione in Django, parte 1

Impariamo attraverso un esempio.

Durante questa guida, ti accompagneremo attraverso la creazione di una semplice applicazione per sondaggi.

Consisterà di due parti:

- Un sito pubblico, che permette agli utenti di visualizzare sondaggi e votarli.
- Un sito amministrativo che ti permette l'aggiunta, modifica ed eliminazione di sondaggi.

Supponiamo che tu abbia già Django installato. Puoi dire che Django sia installato e quale versione sia eseguendo il seguente comando in un prompt della shell (indicato dal prefisso $):

```
...\> py -m django --version
```

Se Django è installato, dovresti vedere la versione della tua installazione. Se non lo è, riceverai un errore che dice «No module named django».

Questo tutorial è scritto per Django 3.2, che supporta Python 3.6 e successivi. Se la versione di Django non corrisponde, puoi consultare il tutorial per la tua versione di Django usando il selettore di versione nell'angolo in basso a destra della pagina, oppure aggiorna Django alla versione più recente. Se stai usando una versione più vecchia di Python, controlla Che versione di Python posso usare con Django? per trovare una versione compatibile di Django.

Vedi Come installare Django per consigli su come rimuovere vecchie versioni di Django ed installarne una più nuova.

Dove trovare aiuto:

Se hai difficoltà a completare questo tutorial, per favore vai alla sezione Getting Help delle FAQ.

Creare un progetto

Se questa è la prima volta volta che usi Django, dovrai prenderti cura di qualche configurazione iniziale. Specificatamente, dovrai auto-generare del codice che costituisce un progetto Django – una collezione di settaggi per un'instanza di Django, che include configurazione del database, opzioni specifiche per Django e settaggi specifici per l'applicazione.

Dalla linea di comando, **cd** dentro la directory dove vorresti tenere il tuo codice, poi chiama il seguente comando:

```
...\> django-admin startproject mysite
```

Questo creerà una directory **mysite** nella directrory presente. Se non funziona, consulta Problemi nell'esecuzione di django-admin.

Nota

Vorrai evitare di chiamare progetti con lo stesso nome di componenti predefiniti in Python o Django. In particolare, questo significa evitare nomi quali **django** (che confligge con Django stesso) o **test** (che confligge con il pacchetto Python di serie).

Dove dovrebbe vivere questo codice?

Se provieni da semplice vecchio PHP (senza uso di framework moderni), probabilmente sei abituato a mettere il codice diremmente sotto la document root del server Web (in un posto quale **/var/www**). Con Django, non fai così. Non è una buona idea mettere alcuno di questo codice Python all'interno della document root del tuo server Web, perché rischi la possibilità che il pubblico possa vedere il tuo codice attraverso il Web. Non è una buona idea per la sicurezza.

Metti il tuo codice in qualche directory **fuori** dalla document root, quale **/home/mycode**.

Diamo un'ochiata a cosa **startproject** ha creato:

```
mysite/
    manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
```

Questi file sono:

- La directory principale **mysite/** è un contenitore per il tuo progetto. Il suo nome non ha importanza per Django; puoi rinominarlo come preferisci.
- **manage.py**: Una utility da linea di comando che ti permette di interagire con questo progetto in svariate maniere. Puoi leggere tutti i dettagli a proposito di **manage.py** in in django-admin and manage.py.
- La cartella interna **mysite/** è il vero pacchetto Python per il tuo progetto. Il suo nome è il nome del pacchetto Python che dovrai usare per importare qualsiasi cosa dall'interno di esso. (es. **mysite.urls**).
- **mysite/__init__.py**: Un file vuoto che indica a Python che questa directory dovrebbe essere considerata un pacchetto Python. Se sei un principiante di Python, leggi ulteriori informazioni sui pacchetti nella documentazione ufficiale di Python.
- **mysite/settings.py**: Settaggi/configurazione per questo progetto Django. Django settings ha tutte le informazioni di cui hai bisogno su come funzionano i settaggi.
- **mysite/urls.py**: The URL declarations for this Django project; a «table of contents» of your Django-powered site. You can read more about URLs in URL dispatcher.
- **mysite/asgi.py**: Un punto di ingresso per i server web compatibili con ASGI per servire il tuo progetto. Vedi How to deploy with ASGI per maggiori dettagli .
- **mysite/wsgi.py**: Un punto d'ingresso per web server compatibili con WSGI per servire il tuo progetto. Vedi How to deploy with WSGI per maggiori dettagli.

Il server di sviluppo

Verifichiamo che il tuo progetto in Django funzioni. Entra nella cartella esterna **mysite**, se non l'hai ancora fatto, e chiama i seguenti comandi:

```
...\> py manage.py runserver
```

Vedrai il seguente output nella linea di comando:

```
Performing system checks...

System check identified no issues (0 silenced).

You have unapplied migrations; your app may not work properly until they are applied.
Run 'python manage.py migrate' to apply them.

novembre 04, 2021 - 15:50:53
Django version 3.2, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Nota

Per ora ignora gli avvisi relativi alle migrazione del database non applicate, ci preoccuperemo a breve del database.

Hai fatto partire il server di sviluppo di Django, un web server leggero scritto interamanete in Python. L'abbiamo incluso in Python così puoi sviluppare cose rapidamente, senza avere a che fare col configurare un server di produzione – quale Apache – finché non sei pronto ad andare in produzione.

Adesso è un buon momento per notare: **non** usare questo server per nulla che somegli ad un ambiente di produzione. È inteso solo per uso durante lo sviluppo. (Il nostro lavoro è fare framework, non server per il Web.)

Ora che il server è attivo apri il tuo browser e vai su http://127.0.0.1:8000/. Troverai una pagina con scritto «Congratulations!» e un razzo che sta decollando. Ha funzionato!

Cambiare la porta

Di default, il comando **`runserver`** avvia il server di sviluppo sull'IP interno alla porta 8000.

Se vuoi cambiare la porta del server, passala come argomento alla linea di comando. Per esempio, questo comando avvia il server sulla porta 8080:

```
...\> py manage.py runserver 8080
```

Se vuoi cambiare l'IP del server, passalo insieme alla porta. Ad esempio, per stare in ascolto su tutti gli IP pubblici disponibili (che è utile se stai eseguendo Vagrant o per mostrare il tuo lavoro su altri computer in rete), usa:

```
...\> py manage.py runserver 0:8000
```

0 è una scorciatoia per **0.0.0.0**. La documentazione completa per il server di sviluppo può essere trovata in **`runserver`**.

Ricaricamento automatico di `runserver`

Quando necessario, il ssrver di sviluppo ricarica automaticamente il codice Python per ogni richiesta. Non c'è bisogno di far ripartire il server per applicare le modifiche al codice. Ciò nonostante, alcune azioni quali l'aggiunta di file non causano un riavvio. In questi casi dovrai quindi riavviare il server.

Creazione dell'applicazione Sondaggi

Ora che il tuo ambiente – un «progetto» – è impostato, sei pronto per iniziare a lavorare.

Ogni applicazione che scrivi in Django è compposta da un pacchetto Python che segue alcune convenzioni. Django fornisce una utility che genera in automatico la struttura di base delle cartelle di una app, così puoi concentrarti sulla scrittura del codice invece che creare le cartelle.

Progetti contro applicazioni

Qual è la differenza tra un progetto e un'app? Un'app è un'applicazione Web che fa qualcosa – es: un sistema di Weblog, un database di record pubblici o una piccola app per sondaggi. Un progetto è una raccolta di configurazioni e app per un determinato sito Web. Un progetto può contenere più app. Un'app può essere in più progetti.

Le tue app possono vivere ovunque sul tuo Python path. In questo tutorial, creeremo la nostra app di sondaggio nella stessa directory del file: **`manage.py`** in modo che possa essere importato come modulo di primo livello, piuttosto che come sottomodulo di **`mysite`**.

Per creare la tua app, sii sicuro di essere nella stessa cartella di **`manage.py`** e scrivi il seguente commando:

```
...\> py manage.py startapp polls
```

Creiamo la nostra cartella **`polls`**,che viene disposta in questo modo

```
polls/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    views.py
```

La struttura di questa cartella contiene la nostra applicazione sondaggio.

Scrivi la tua prima vista

Scriviamo la nostra prima vista. Apri il file **`polls/views.py`** e scrivi il seguente codice Python:

polls/views.py

```
from django.http import HttpResponse


def index(request):
    return HttpResponse("Hello, world. You're at the polls index.")
```

Questa è il metodo più semplice di scrivere una vista in Django. Per fare una chiamata alla vista, dobbiamo mapparla in una URL - e per questo abbiamo bisogno di URLconf.

Per creare una URLconf nella cartella polls, crea un file col nome **`urls.py`**. La cartella della tua app ora dovrebbe essere come:

```
polls/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    urls.py
    views.py
```

Nel file **`polls/urls.py`** includi il seguente codice:

polls/urls.py

```
from django.urls import path

from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

Il passo successivo è puntare l'URLconf principale al modulo **`polls.urls`**. In **`mysite/urls.py`**, aggiungi un'importazione per **`django.urls.include`** e inserisci un **`include()`** nell'elenco **`urlpatterns`**, in modo da avere:

mysite/urls.py

```
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('polls/', include('polls.urls')),
    path('admin/', admin.site.urls),
]
```

La funzione **`include()`** permette di fare riferimento ad altre URLconf. Ogni volta che Django incontra **`include()`**, rimuove qualsiasi parte dell'URL combaciata fino a quel punto e invia la stringa rimanente alla URLconf inclusa per un'ulteriore elaborazione.

L'idea alla base di **`include()`** è di semplificare il plug-and-play deglle URL. Poiché i sondaggi si trovano nella propria URLconf (**`polls/urls.py`**), possono essere posizionati dentro «/polls/», o dentro «/fun_polls/», o dentro «/content/polls/», o qualsiasi altro percorso e l'app continuerà a funzionare.

Quando usare `include()`

Dovresti usare sempre **`include()`** quando includi altri pattern URL. L'unica eccezione a questo è **`admin.site.urls`**.

Ora hai collegato una vista **`index`** nella URLconf. Verifica che funzioni con il seguente comando:

```
...\> py manage.py runserver
```

Nel tuo browser vai all'indirizzo http://localhost:8000/polls/ , e dovresti vedere il testo «*Hello, world. You're at the polls index.*», che hai definito nella view **`index`**.

Non hai trovato la pagina?

Se ottieni una pagina di errore, controlla di stare andando su http://localhost:8000/polls/ e non http://localhost:8000/.

Alla funzione **`path()`** vengono passati quattro argomenti, due obbligatori: **`route`** e **`view`** e due opzionali: *kwargs*`, and **`name`**. A questo punto, vale la pena rivedere a cosa servono questi argomenti.

Argomento di `path()`: `route`

`route` è una stringa che contiene un schema di URL. Durante l'elaborazione di una richiesta, Django inizia dal primo schema in **`urlpatterns`** e si fa strada lungo l'elenco, confrontando la URL richiesta con ogni schema fino a quando non trova quello corrispondente.

I pattern non cercano i parametri GET e POST o il nome di dominio. Ad esempio, in una richiesta a **`https://www.example.com/myapp/`**, URLconf cercherà **`myapp/`**. In una richiesta a *https://www.example.com/myapp/?page=3*`, URLconf cercherà anche **`myapp/`**.

Argomento di `path()`: `view`

Quando Django trova uno schema di corrispondente, chiama la funzione di visualizzazione specificata con un oggetto di tipo **`HttpRequest`** come primo argomento e qualsiasi valore «catturato» dal percorso come argomenti chiave. Ne faremo un esempio tra poco.

Argomento di `path()`: `kwargs`

Argomenti arbitrari di parole chiave possono essere passati in un dizionario alla vista di destinazione. Non useremo questa funzionalità di Django nel tutorial.

Argomento di `path()`: `name`

Assegnare un nome alla tua URL ti consente di fare riferimento ad esso in modo inequivocabile da qualsiasi altra parte in Django, specialmente all'interno dei template. Questa potente funzionalità ti consente di apportare modifiche globali al modello delle URL del tuo progetto toccando solo un singolo file.

Quando sei a tuo agio con il flusso base di richiesta e risposta, leggi parte 2 di questo tutorial per iniziare a lavorare con il database.

Guida rapida di installazione

Scrivere la tua prima applicazione in Django, parte 2

Get Involved

Join a Group

Contribute to Django

Submit a Bug

Report a Security Issue

Follow Us

GitHub

Twitter

News RSS

Django Users Mailing List

Support Us

Sponsor Django

Official merchandise store

Amazon Smile

Benevity Workplace Giving Program