
django

introduzione

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

introduzione

django è un ***web framework*** "***free***" e ***open-source*** basato su Python che utilizza il ***pattern architetturale model-template-view***…

...cosa significa?
"***free***" → in questo caso con l'accezione di "free as in beer", gratuito;

open-source → il codice è pubblico e modificabile da chiunque;

framework → letteralmente struttura, intelaiatura.
In informatica, un software framework è un'applicazione "vuota", una "bozza" che può essere facilmente personalizzata in una serie di specifiche applicazioni reali.
Nel caso di un web framework, il fine ultimo è quello di creare applicazioni web, offrendo allo sviluppatore strumenti modulari (flessibili, personalizzabili e riutilizzabili) per tutte quelle che sono le funzionalità basilari (e quasi sempre presenti) nella realizzazione di un'applicazione web (autenticare e registrare gli utenti, connettersi e scrivere su un database, servire contenuto HTML, etc…).

pattern architetturale → una particolare struttura "architettonica" che sottende all'intero framework, una struttura che impone un certo rigore nel workflow del lavoro. Nel caso di django, il pattern architetturale è **model**(definizione di tabelle sul database)-**template**(pagina HTML)-**view**(business logic e gestione delle risposte del server), concetti che avremo modo in approfondire.

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

introduzione

Quindi:

django è un framework gratuito e open-source che offre una collezione di strumenti per creare applicazioni web utilizzando la struttura model-template-view

Perché usarlo?

- Veloce → è pensato per realizzare applicazioni il più velocemente possibile
- Completo → offre dozzine di strumenti per svolgere le funzionalità più comuni
- Sicuro → aiuta gli sviluppatori ad evitare tipici errori legati alla sicurezza
- Scalabile → capace di gestire anche il traffico di piattaforme con migliaia di utenti
- Versatile → in grado di gestire dai sistemi di gestione dei contenuti, ai social network alle piattaforme di computazione scientifica

La versione *stable* più recente è la 2.2.6.

Per informazioni sull'installazione
https://docs.djangoproject.com/en/2.2/intro/install/

pipenv

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

pipenv

Il *virtual environment* (ambiente virtuale) è uno strumento che mantiene separate le dipendenze richieste da diversi progetti creando per loro ambienti virtuali isolati. Per lavorare sul nostro primo progetto utilizzeremo un *virtual environment* creato con pipenv nel quale installeremo django.

1 - Per prima cosa creiamo una cartella e accediamo alla stessa

1
```
$ mkdir tutorial_project
$ cd tutorial_project/
```

2 - Accertiamoci di avere installato pipenv.
Se così non fosse, è sufficiente eseguire il comando
$ pip3 install pipenv

3 - installiamo django nel nostro *virtual environment*.

2
```
$ pipenv --version
pipenv, version 2018.11.26
```
3
```
$ pipenv install django
Creating a virtualenv for this project…
```

4 - Attiviamo il *virtual environment*

4
```
$ pipenv shell
Launching subshell in virtual environment…
```

5 - Il prompt riporterà tra parentesi il nome del *virtual environment* utilizzato

5
```
(tutorial_project)$
```

startproject

startproject

django-admin è il comando di django per le attività amministrative.

Seguito dall'opzione **startproject** e dall'argument **<nome_progetto>** crea un nuovo progetto django.

Esegui:
```
django-admin startproject tutorial ./
```

Eseguito questo comando, la cartella *tutorial_project* sarà così strutturata:

```
tutorial_project/
    tutorial/
        __init__.py
        settings.py
        urls.py
        wsgi.py
    manage.py
    Pipfile
    Pipfile.lock
```

settings.py controlla i setting del nostro progetto;
urls.py gestisce quello che django dovrà restituire a seguito di una richiesta;
wsgi.py aiuta django a servire eventuali pagine web;
manage.py è usato per eseguire diversi comandi di django come eseguire un web server locale o creare una nuova applicazione.

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

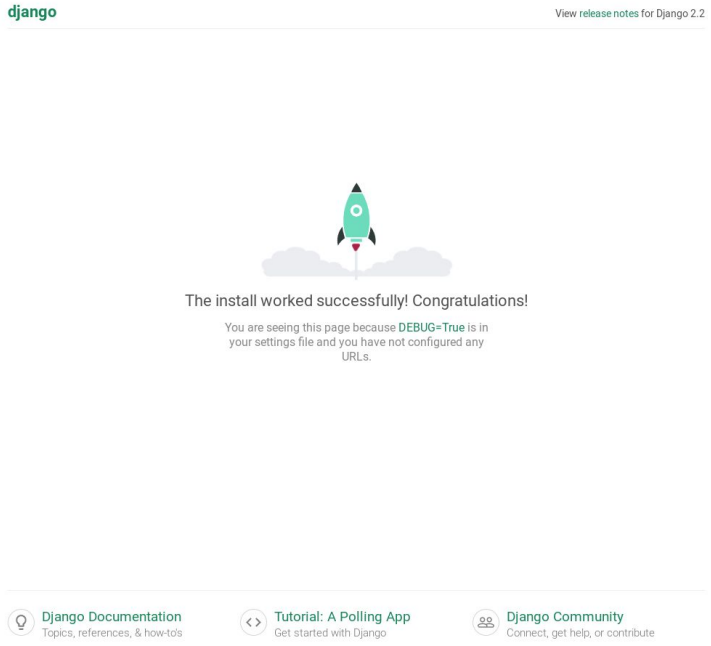
startproject

django è dotato di un web server atto allo sviluppo locale.

È possibile lanciarlo con in comando **`runserver`** (dalla cartella dove si trova manage.py)

Esegui:
`python manage.py runserver`

Dopo l'esecuzione del comando, tra le varie informazioni stampate sul terminale troviamo anche
`Starting development server at http://127.0.0.1:8000/`
Seguendo questo link, accediamo alla pagina rappresentata a fianco.



UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

settings.py

static files → file statici aggiuntivi alla pagina web come immagini, javascript, css, etc…

collectstatic → comando per aggregare tutti gli static file provenienti dalle app del progetto in un'unica cartella. *(viene utilizzato per eseguire deployment ottimizzati)*

BASE_DIR → path assoluta da utilizzare in TEMPLATES, STATIC_ROOT, etc...

TEMPLATES → definisce, tra le altre cose, la location dei *template* del progetto.

STATIC_ROOT → la path assoluta alla cartella nella quale collectstatic aggiunge static file.

STATIC_URL → url da utilizzare per accedere agli static file presenti in STATIC_ROOT.

STATICFILES_DIRS → le varie path relative agli static files provenienti dalle varie app.

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

startproject

django utilizza il concetto di “app” e “project” per mantenere il codice pulito, leggibile e modulare.

Quali sono le differenze tra app e project?

Un’app è un’applicazione web che ha funzionalità particolari e specifiche - es. un sistema Weblog, un database di record pubblici o una semplice app per sondaggi.

Un project è una collezione di configurazioni e app per un’applicazione web specifica.

Un’app può essere utilizzata in molteplici project.

settings.py

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

settings.py

settings.py è un file che contiene tutte le configurazioni del progetto django. Molte delle opzioni contenute al suo interno possono risultare oscure la prima volta che si incontrano.

Per utilizzare le funzionalità più basilari di django è però sufficiente saper gestire alcune opzioni specifiche di settings.py, tra queste:

DEBUG → in caso di errori, con valore True mostra su schermo l'errore nel dettaglio. Con valore False si limita a dichiarare lo "status code" della risposta. True si utilizza in fase di sviluppo, False in produzione;

ALLOWED_HOST → il suo scopo è convalidare l'header host HTTP di una richiesta. È un'opzione strettamente legata a DEBUG (con DEBUG=False e ALLOWED_HOST=[] django non si avvia) e alla fase di produzione;

INSTALLED_APPS → qui si inseriscono tutte le app presenti nel progetto; alcune di queste sono app e librerie di base di django altre saranno invece scritte da noi;

templates

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

templates

Essendo un framework web, Django ha bisogno di un modo conveniente per generare HTML in modo dinamico. L'approccio più comune si basa sui *templates*. Un *template* contiene le parti statiche dell'output HTML desiderato, nonché alcune sintassi speciali che descrivono come verranno inseriti i contenuti dinamici.

Cos'è un template?

In django un template (letteralmente, "modello") è un file HTML con contenuto statico e dinamico.

In settings.py assegnamo alla chiave DIRS nel dizionario all'interno di TEMPLATES il seguente codice:

```
os.path.join(BASE_DIR, 'templates')
```

e creiamo una cartella templates all'interno del nostro progetto.

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

templates

django utilizza un linguaggio per i template che permette di specificare come rappresentare i dati.
È basato su *template tags*, *template variables* e *template filters*.

- I ***template tags*** controllano il rendering dei template:
  `{% tag %}`
- Le ***template variables*** sono rimpiazzate da valori quando il template viene "renderizzato":
  `{{ variable }}`
- I ***template filters*** permettono di modificare variabili nel momento del rendering del template:
  `{{ variable|filter }}`

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

templates

Come già accennato, i *template tag* sono utili per controllare la rappresentazione dei template e la composizione degli stessi anche attraverso la combinazione di più template:

{% block <block_name> **%}** → questa tag dice a django che stiamo definendo un blocco di contenuto;

{% extends <"template_name"> **%}** → questa tag si utilizza per comporre template insieme.

Nell'ambito dello sviluppo web capita spesso che alcuni elementi debbano comparire su più pagine.
Nell'esempio a fianco abbiamo un file chiamato **base.html** (il nome "base" è una convenzione in django, ma non obbligatorio).
Al suo interno abbiamo definito una tag {% block %} con il nome **title** e una con il nome **content**.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>{% block title %}{% endblock %}</title>
</head>
<body>
  <!-- NAVBAR -->
  <section class="container-fluid h-100 pt-5">
  {% block content %}

  {% endblock %}
</body>
</html>
```

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

templates

Ora “estendiamo” il file **base.html** sul file **home.html**. In questo modo, tutto il contenuto di **base.html** verrà renderizzato in **home.html** (DOCTYPE, <html>, <head>, <body>, navbar, </body>, </html>).
Così sarà sufficiente inserire del contenuto delle tag {% block %} in home.html per realizzare una pagina html completa.

```
{% extends 'base.html' %}

{% block title %}Home page{% endblock %}

{%  block content %}
<h1>La mia home page</h1>
<p>
  Lorem ipsum dolor sit amet, consectetur adipisicing elit,
  sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
</p>

{% endblock %}
```

Risultato finale:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Home page</title>
</head>

<body>
  <!-- NAVBAR -->
  <section class="container-fluid h-100 pt-5">
    <nav class="navbar fixed-top navbar-expand-lg navbar-light">
      <a class="navbar-brand h-100" href="#">
        <h3 id="logo">Logo</h3>
      </a>
    </nav>
  </section>

<h1>La mia home page</h1>
<p>
  Lorem ipsum dolor sit amet, consectetur adipisicing elit,
  sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
</p>


</body>

</html>
```

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

templates

{% load static %} è una tag che dice a django di caricare le **static tag**.
In questo modo è possibile utilizzare la sintassi **{% static** *'path'* **%}** per accedere agli static file presenti nella cartella **static**.

Template syntax

```
<img src="{% static 'img/carousel_1.jpg' %}">
```

path

```
tutorial_project/
        tutorial/
        static/
                img/
                        carousel_1.jpg
        [...]
```

settings.py

```
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
STATIC_URL = '/static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
```

views.py

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

views.py

views.py è un file costituito da *view* (viste). Ogni *view* è responsabile della business logic e di cosa il server risponde a seguito di una particolare richiesta ritornando all'utente un oggetto **HTTPResponse** o sollevando un'eccezione come HTTP404.

Creando un project, il file views.py non viene generato automaticamente.

Nella cartella tutorial creiamo un file views.py nel quale inseriremo il seguente codice:

```
from django.http import HttpResponse


def home(request):
    return HttpResponse("Hello world! Questa è la home page.")
```

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

views.py

Esistono molte funzioni e classi pensate per ritornare un oggetto HTTPResponse e mostrare su schermo un template.

Tra gli strumenti più usati:
render() → *function*. Combina un template con dei parametri passati tramite l'oggetto context.
TemplateView → *class*. Mostra su schermo uno specifico template con il context contenente i parametri presenti nell'url.

Per mostrare dati:
ListView → *class*. self.object_list* conterrà la lista degli oggetti (di solito, ma non necessariamente, un queryset) sul quale la view sta operando;
DetailView → *class*. self.object* conterrà l'oggetto sul quale la view sta operando.

context → un dizionario che mappa coppie chiave/valore che viene utilizzato nei template per generare contenuto dinamico.

* attributo accessibile dal template

urls.py

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

urls.py

Nel file urls.py è presente una lista nella quale ciascun elemento rappresenta un url collegato ad una *view* specifica (funzione o classe che sia) e a un nome che può essere utilizzato nei template.

Nel file urls.py all'interno del nostro progetto inseriamo il seguente codice:

```
from django.contrib import admin
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('admin/', admin.site.urls),
]
```

Dalla stessa directory dove si trova il file urls.py, importa il file views.py

path(*route*, *view*, *name***)**:
'' → stringa vuota. Si accede a questa view senza aggiungere nessun argument all'url;
views.home → esegue la funzione *home* del file views.py
'**home'** → assegna il nome 'home' alla path (per utilizzare l'*url reversing*).

francesco.faenza@unimore.it