Django

Prendiamo confidenza

Django

Abbiamo visto cosa comporta creare un progetto in django.

Abbiamo fatto una rapida carrellata di diversi argomenti correlati alla struttura di files e directory che appaiono “magicamente” alla creazione di un progetto

E’ ora di prendere confidenza con django, facendo qualche prova

Installazione nuovo progetto

abbiamo visto gli step necessari

- Impostazione di un ambiente virtuale tramite pipenv
- Installazione nell'ambiente virtuale di eventuali librerie "accessorie"
- *django-admin startproject* **nomeprogetto**
- …

Promemoria

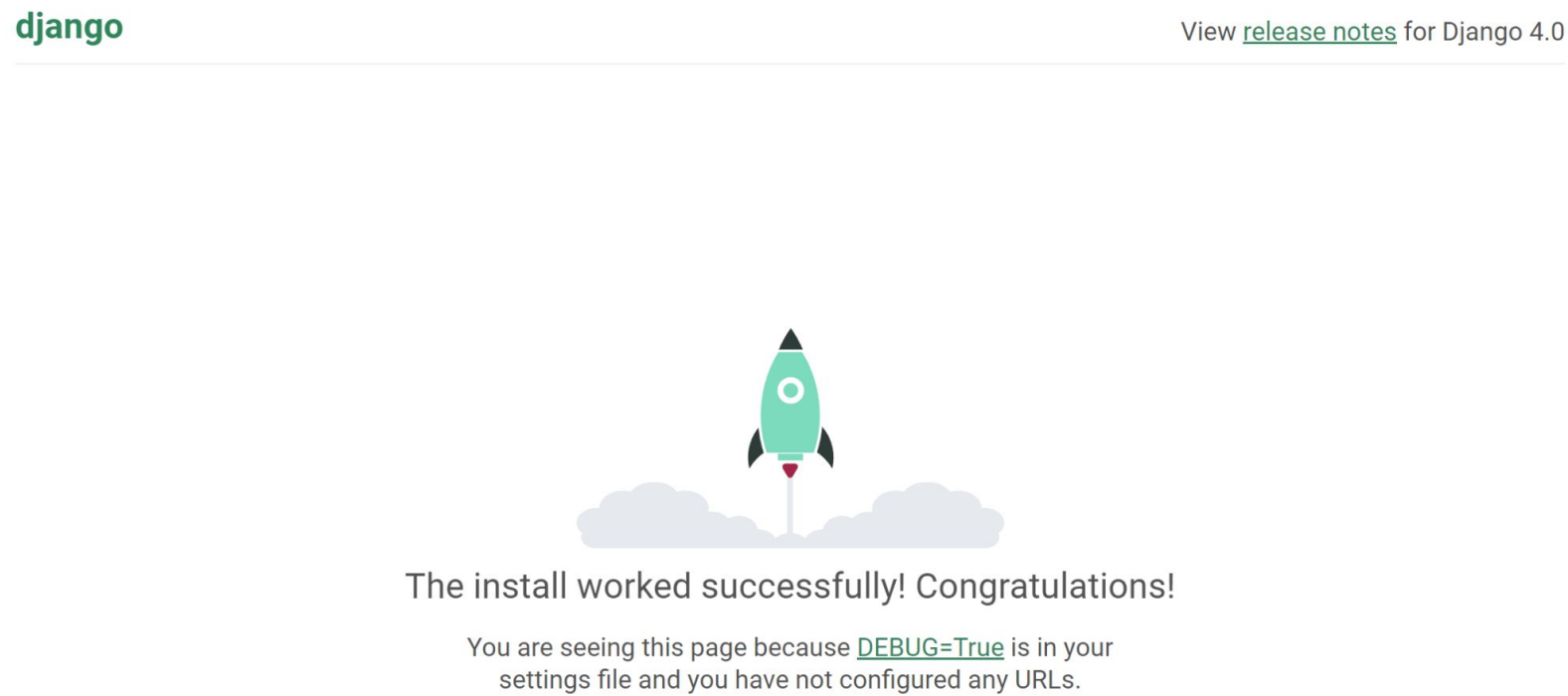
Una volta creato l’ambiente virtuale ed installato django con *pipenv install django*:

assicuratevi tramite *pipenv shell* ed in seguito *pipenv --venv* di essere difatto all’interno dell’ambiente virtuale.

Per uscire da un ambiente virtuale:

in cmd line: *exit*

python manage.py runserver



Su un qualsivoglia browser puntato su 127.0.0.1:8000

Se non lo vedessi?

Può succedere. Del resto:

> **python.exe .\manage.py runserver**
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
February 10, 2022 - 15:51:51
Django version 4.0.2, using settings 'primo_progetto.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

Altro test di funzionamento

da riga di comando

>python

Questo attiva una shell di python

>import django

Se non ci sono errori, abbiamo comunque fatto tutto correttamente.

L’eventuale errore sulle migrazioni verrà risolto in seguito.

urls

In urls.py e creando il file views.py

Diamo un messaggio di benvenuto ai nostri visitatori…

- creazione del file views.py
- implementazione di una funzione che prende in ingresso un richiesta e restituisce un oggetto di tipo *HttpResponse*
- Inserimento dell'url che punta alla funzione così definita
- Provare il tutto portandosi su tale url

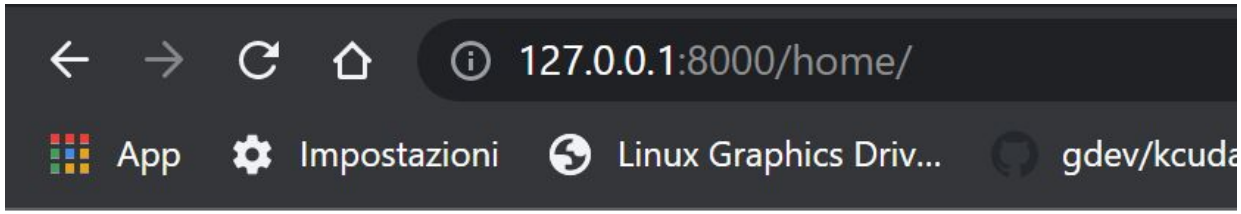
Implementazione

primo_progetto > views.py > home_page

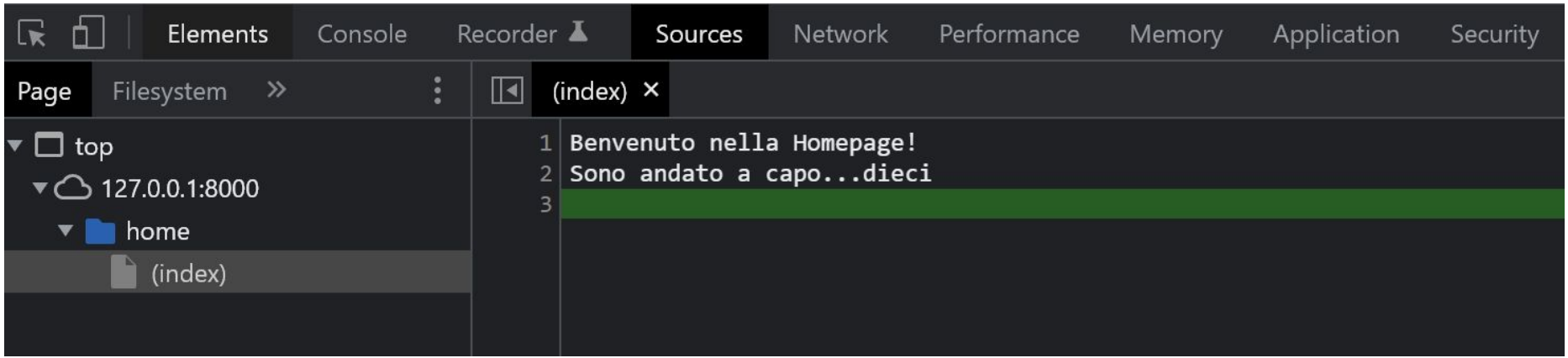
```python
from django.http import HttpResponse


def home_page(request):

    response = "Benvenuto nella Homepage!\n"
    response += "Sono andato a capo...dieci\n"

    return HttpResponse(response)
```

```python
#siamo in primo_progetto/urls.py
from django.contrib import admin
from django.urls import path
from .views import home_page

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

Risultato (127.0.0.1:8000/home/)



Benvenuto nella Homepage! Sono andato a capo...dieci



Cosa abbiamo imparato?

Abbiamo imparato cosa succede quando cerchiamo di **fare render** di una stringa su un browser che si aspetterebbe una pagina html o formati simili.

Da qui l’importanza del concetto di template, che dettaglieremo in seguito.

Senza template, il lato presentazione della nostra web app si riduce ad un lettore di informazioni testuali, neanche fatto tanto bene (i caratteri new line sono stati sostanzialmente ignorati)

Altre sorprese?

Si.

Torniamo su 127.0.0.1:8000…

Si è rotto…?

Page not found (404)

Request Method: GET
Request URL: http://127.0.0.1:8000/

Using the URLconf defined in `primo_progetto.urls`, Django tried these URL patterns, in this order:

1. `admin/`
2. `home/ [name='homepage']`

The empty path didn't match any of these.

You're seeing this error because you have `DEBUG = True` in your Django settings file. Change that to `False`, and Django will display a standard 404 page.

No. Semplicemente, non appena editiamo il file urls.py, la schermata di benvenuto data in "regalo" da django non esiste più…

In compenso otteniamo una serie di informazioni utili come debug dump

Creiamo un alias

Vorrei fare in modo che più indirizzi puntino alla stessa home page che ho appena implementato.

Soluzione poco intelligente:

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

E la soluzione più intelligente?

Possiamo creare dei pattern che risolvono alla stessa pagina.

Ci sono diversi modi per farlo.

Il modo più sintetico ed elegante è tramite l’utilizzo di regular expressions (regex)

Python ha un motore di regex molto maturo e django è in grado di sfruttarlo per diverse cose, tra cui la generazione di string per pattern matching sui percorsi delle pagine web create dinamicamente.

Regex

Un'espressione regolare (regular expression o regex) è una sequenza di caratteri che identifica tramite un meccanismo di pattern matching un insieme di stringhe.

In parole povere, una regex stabilisce una serie di regole che una stringa deve seguire al fine di fare “match” con i\il pattern descritto\i dalla regex stessa.

Un motore di regex prende in ingresso un’espressione regolare e una stringa da controllare e restituisce se esiste una corrispondenza

re_path in django urls

```
#siamo in primo_progetto/urls.py
from django.contrib import admin
from django.urls import path, re_path
from .views import home_page

urlpatterns = [
    path('admin/', admin.site.urls),
    re_path(r"^$|^/$|^home/$",home_page,name="homepage")

    #path("home/", home_page, name="homepage"),
    #path("/", home_page, name="homepage"),
    #path("",home_page,name="homepage")

]
```

Cosa significa

Le regex introducono qualcosa di simile ad un linguaggio dentro un linguaggio.

https://cheatography.com/davechild/cheat-sheets/regular-expressions/

```
r"^$|^/$|^home/$"
```

r : specifica che la stringa che segue è una regex.
^ : inizio stringa
$: fine stringa
| : Alternativa

Più nel dettaglio

```
r"^$|^/$|^home/$"
```

- Alternativa 1:
  - Nessun carattera tra inizio (^) e fine ($) stringa
- Alternativa 2:
  - Il carattere ammesso (la barra ‘/’) tra inizio e fine stringa
- Alternativa 3:
  - La parola ammessa, compresa tra inizio e fine stringa è ‘home/’

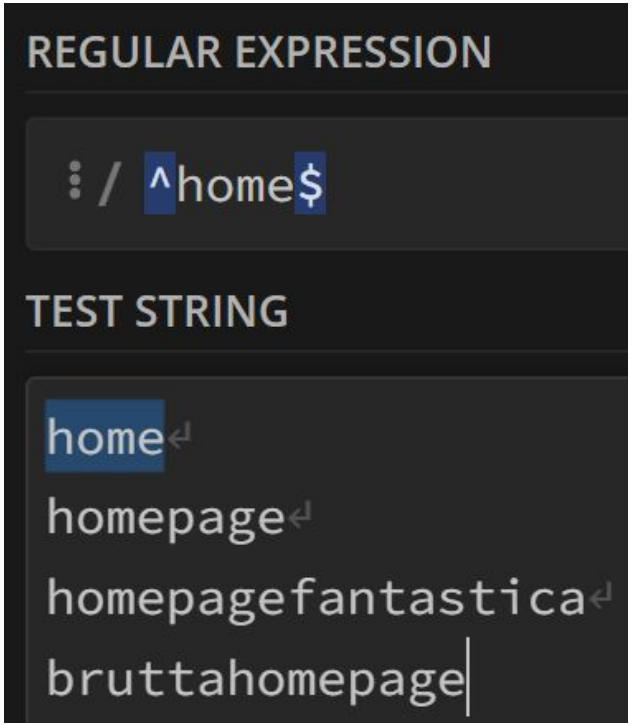
Che succede se…?

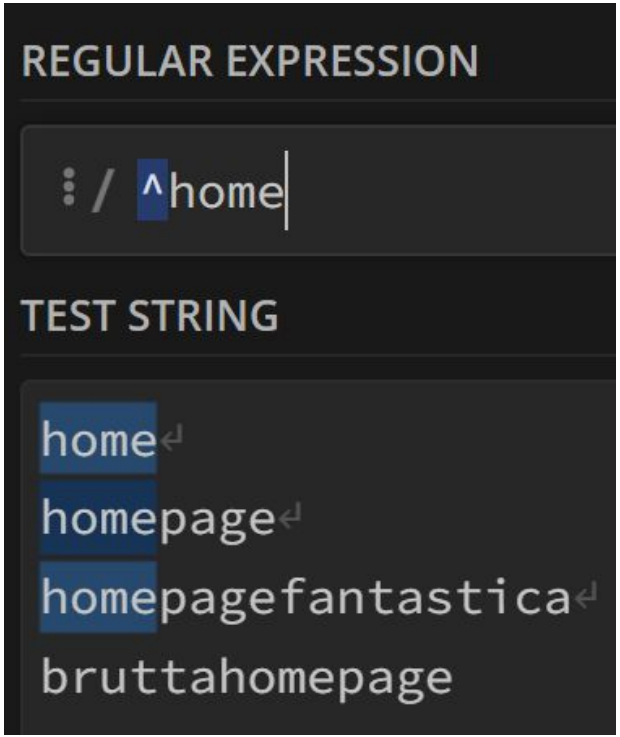
```
r"^$|^/$|^home"
```

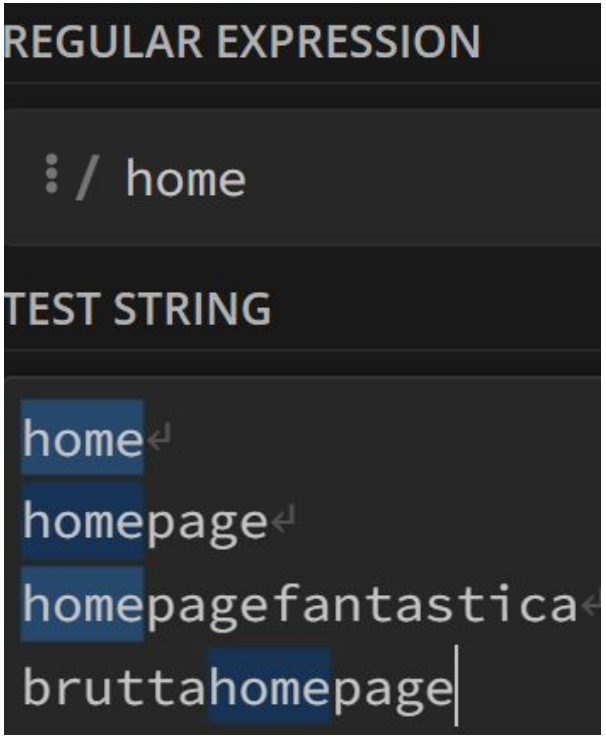
Ho leggermente modificato la terza alternativa.

Possiamo provare anche su https://regex101.com/

Proviamo

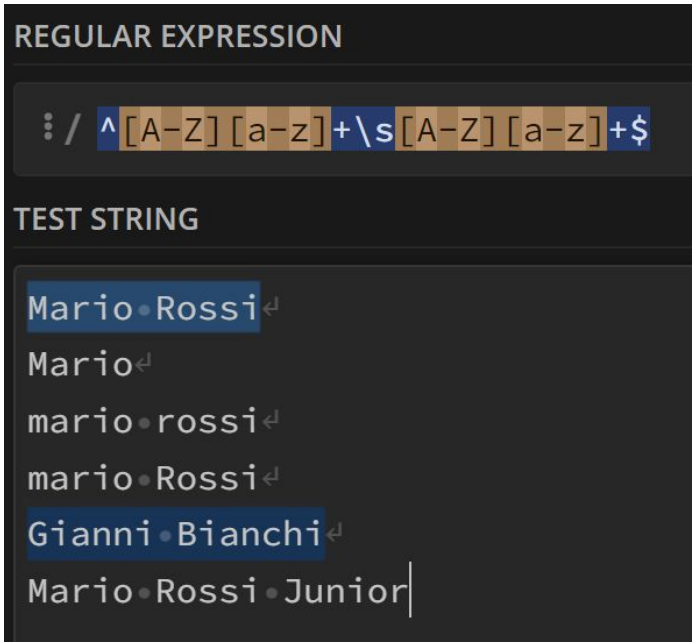

REGULAR EXPRESSION
/ ^home$
TEST STRING
home
homepage
homepagefantastica
bruttahomepage


REGULAR EXPRESSION
/ ^home
TEST STRING
home
homepage
homepagefantastica
bruttahomepage


REGULAR EXPRESSION
/ home
TEST STRING
home
homepage
homepagefantastica
bruttahomepage

Cos’altro possiamo fare con le regex?

Controllo input dell’utente per costringerlo ad inserire stringhe ben formattate.



Torniamo alla nostra view

C’è un aspetto che abbiamo momentaneamente ignorato:

```
def home_page(request):
    …
```

Che cosa ne facciamo della nostra request? Proviamo a stamparla per capire come è fatta.

Printing & logging con Django

print() funziona

Anche se si dovrebbero usare le funzioni di logging.

print(request)

in **home_page**, aggiungiamo

print*(“RESPONSE: ” + str(request))*

*RESPONSE: <WSGIRequest: **GET** '/home/'>*

Cos’altro ci dice?

print("Caratteristiche di request " + **str**(**dir**(request)))

Caratteristiche di request ['COOKIES', 'FILES', 'GET', 'META', 'POST', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_current_scheme_host', '_encoding', '_get_full_path', '_get_post', '_get_raw_host', '_get_scheme', '_initialize_handlers', '_load_post_and_files', '_mark_post_parse_error', '_messages', '_read_started', '_set_content_type_params', '_set_post', '_stream', '_upload_handlers', 'accepted_types', 'accepts', 'body', 'build_absolute_uri', 'close', 'content_params', 'content_type', 'csrf_processing_done', 'encoding', 'environ', 'get_full_path', 'get_full_path_info', 'get_host', 'get_port', 'get_signed_cookie', 'headers', 'is_secure', 'method', 'parse_file_upload', 'path', 'path_info', 'read', 'readline', 'readlines', 'resolver_match', 'scheme', 'session', 'upload_handlers', 'user']

Evidenziando gli attributi?

```
for e in request.__dict__:
    print(e)
print("USER " + str(request.user))
print("PATH " + str(request.path))
```

environ
path_info
path
META
method
content_type
content_params
_stream
_read_started
resolver_match
COOKIES
session
user
_messages
csrf_processing_done
USER AnonymousUser
PATH /home/

N.B. per accedere a `request.user` occorre eseguire il comando *`python manage.py migrate`*
Ulteriori dettagli in seguito…

Logging in Django

Django di suo ha già un sistema di log molto complesso che di default è attivo.

Pertanto, se si intende modificare o integrare il logger pre-esistente in django, occorre impostare una serie di parametri in settings.py

https://docs.djangoproject.com/en/4.0/topics/logging/

Faremo comunque una prova

```python
from django.http import HttpResponse

import logging
logger = logging.getLogger(__name__)


def home_page(request):

    response = "Benvenuto nella Homepage, " + str(request.user)


    """
    print("RESPONSE " + str(request))
    print("Caratteristiche di request " + str(dir(request)))

    for e in request.__dict__:
        print(e)

    print("USER " + str(request.user))
    print("PATH " + str(request.path))
    """

    if not request.user.is_authenticated:
        logger.warning(str(request.user) + " non è autenticato!")

    return HttpResponse(response)
```

logging.<severità del messaggio>(“messaggio”)

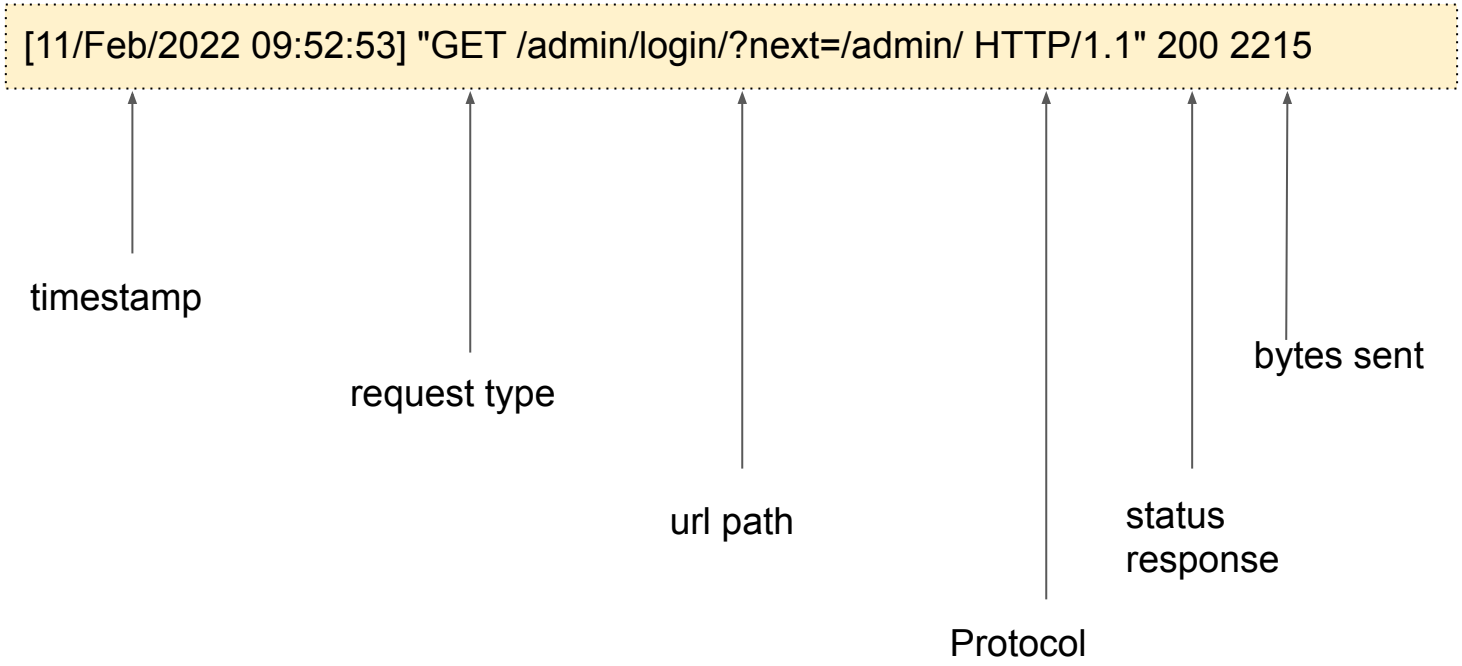
Le impostazioni di logging su settings.py vanno aggiunte from scratch.

tra le impostazioni del logger in settings.py è possibile stabilire la “severità” oltre la quale i messaggi di logging finiscono in stdout

Logging di default di Django

Lo si vede nella console in cui abbiamo dato il comando *runserver*

```
[11/Feb/2022 09:52:53] "GET /admin HTTP/1.1" 301 0
[11/Feb/2022 09:52:53] "GET /admin/ HTTP/1.1" 302 0
[11/Feb/2022 09:52:53] "GET /admin/login/?next=/admin/ HTTP/1.1" 200 2215
[11/Feb/2022 09:52:53] "GET /static/admin/css/nav_sidebar.css HTTP/1.1" 200 2616
[11/Feb/2022 09:52:53] "GET /static/admin/css/base.css HTTP/1.1" 200 19513
[11/Feb/2022 09:52:53] "GET /static/admin/css/login.css HTTP/1.1" 200 954
[11/Feb/2022 09:52:53] "GET /static/admin/css/responsive.css HTTP/1.1" 200 18575
[11/Feb/2022 09:52:53] "GET /static/admin/js/nav_sidebar.js HTTP/1.1" 200 3401
[11/Feb/2022 09:52:53] "GET /static/admin/css/fonts.css HTTP/1.1" 200 423
[11/Feb/2022 09:52:53] "GET /static/admin/fonts/Roboto-Light-webfont.woff HTTP/1.1" 200 85692
[11/Feb/2022 09:52:53] "GET /static/admin/fonts/Roboto-Regular-webfont.woff HTTP/1.1" 200 85876
[11/Feb/2022 09:53:23] "GET /welcome_user HTTP/1.1" 200 10
```


[11/Feb/2022 09:52:53] "GET /admin/login/?next=/admin/ HTTP/1.1" 200 2215
timestamp
request type
url path
Protocol
status response
bytes sent

Http Response Codes (in breve)

1XX	Messaggio informativo
2XX	Success!
3XX	Redirection
4XX	Fail (colpa del client…)
5XX	Fail (colpa del server…)

https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

4XX vs 5XX

Per “scatenare” un **404**, mi basta inserire da browser un url non “matchabile” con le regole elencate in *urlpatterns*

```
[11/Feb/2022 10:59:26] "GET /paginainesistente/ HTTP/1.1" 404 2447
```

Per “scatenare” un **500**, mi basta causare un errore lato server, e.g. un syntax error in python

```
    [....precede stack trace]
a = int(request)
TypeError: int() argument must be a string, a bytes-like object or a number, not 'WSGIRequest'
[11/Feb/2022 11:01:47] "GET /welcome_user HTTP/1.1" 500 61480
```

4XX e 5XX in DEBUG\RELEASE

E’ possibile **in release** fare in modo che django “intercetti” queste risposte fornendo all’utente pagine personalizzate, anziché il default render del browser.

In **debug** conviene invece **leggere attentamente** le informazioni che compaiono in caso di errori. Ci aiutano a correggere eventuali errori.

Ma forse parlano troppo….?

Torniamo un attimo indietro: richieste GET

in **home_page**, aggiungiamo

print(*“RESPONSE: ” + str(request))*

RESPONSE: <WSGIRequest: ***GET*** *'/home/'>*

Le richieste get ammettono parametri specificati tramite URL

Passaggio di parametri tramite GET params

*url_path**?param1=value1&param2=value2…***

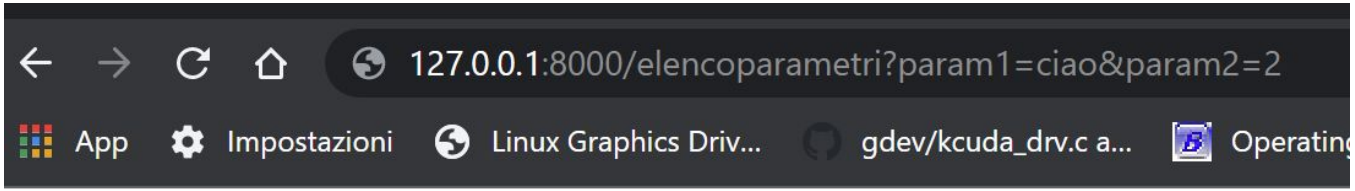
nella mia view function, ottengo questi parametri tramite *request.GET[“param_name”]*

Esempio

in urls.py
tra gli urlpatterns:
***path**("elencoparametri/",elenca_params,name="params")*

La funzione “elenca_params” dovrà quindi essere implementata ed importata in views.py:

```
def elenca_params(request):

    response = ""
    for k in request.GET:
        response += request.GET[k] + " "

    return HttpResponse(response)
```



Esercizi di esempio:

- Si faccia una view che risponde ad un url che tramite richiesta GET manda il parametro *num*. Se è un numero intero, la pagina web generata informa l'utente se il numero richiesto è pari o dispari, altrimenti scrive "Non è un numero"
- Si faccia una view che risponde ad un url che tramite richiesta GET manda il parametro *nome*. Supponendo tale parametro sia il nome del visitatore, lo si saluti: "Ciao <nome>"
- Si faccia una view che risponda ad una richiesta GET, senza parametri. L'url deve essere in formato *localhost:8000/welcome_<nome>/* che sia quindi in grado di salutare l'utente in funzione della sottostringa <nome>.

Riassumendo

Abbiamo imparato l’uso di urlpatterns in urls.py e relative view(s) di risposta.

Abbiamo visto come non solo il contenuto della pagina possa essere dinamico, bensì anche l’indirizzo stesso possa essere “generato” al volo.

Abbiamo visto come sia possibile passare una serie arbitraria di parametri tramite i campi delle richiesta GET.

Altro modo per passare parametri

Usare i parametri GET per passare argomenti alle nostre views è uno dei tanti approcci possibili.

Esistono altre richieste, e.g. POST che vedremo in seguito

Rimane il fatto che la generazione della stringa *<url_path>?p=v&....*

non viene solitamente “scritta” dagli utenti, ma il risultato di un’azione client\webserver side. Questo perchè non è molto “user friendly”

Esiste un altro metodo ancora…

Passaggio di parametri tramite URL path

E’ possibile inserire in urlpatterns un’espressione di questo tipo:

path**('url_path/<**int**:eta>/', view_func, name='alias') **#un solo param di tipo int chiamato “eta”

path**('url_path/<**str:**nome>/<**int**:eta>/', view_func, name='alias') **#2 params…

Cosa cambia nell’url?

Sarà ora possibile raggiungere tutte le combinazioni valide imposte dalle regole espresse in urlpatterns, esempi:

- localhost:8000/url_path/Mario/23/
- localhost:8000/url_path/Carlo/18/
- …

Cosa cambia nella view function?

Riesco ad accedere a tali parametri come argomenti in ingresso alla mia funzione:

```
def welcome_path(request,nome,eta):
    [....]
```

Proviamo

```
urlpatterns = [
    path('admin/', admin.site.urls),
    re_path(r"^welcome_[A-Za-z0-9]+$",welcome_user,name="welcomeuser"),
    path("welcome_path/<str:nome>/<int:eta>/",welcome_path,name="welcomepath")
]
```

```
def welcome_path(request,nome,eta):
    return HttpResponse("Si chiama " + nome + " ed ha " + str(eta) + " anni")
```

Si chiama Mario ed ha 23 anni

Proviamo a “romperlo”

- *http://127.0.0.1:8000/welcome_path/Mario Rossi/23/*
- *http://127.0.0.1:8000/welcome_path/5/23/*
- *http://127.0.0.1:8000/welcome_path/Mario/Ciao/*
- *http://127.0.0.1:8000/welcome_path/Mario/2.3/*

Quali di questi “rompe” il giochino e solleva un 404?

Proviamo a “romperlo”

- *http://127.0.0.1:8000/welcome_path/Mario Rossi/23/*
  - *OK: http://127.0.0.1:8000/welcome_path/Mario**%20**Rossi/23/*
- *http://127.0.0.1:8000/welcome_path/5/23/*
  - *OK*
- *http://127.0.0.1:8000/welcome_path/Mario/Ciao/*
  - *KO: “Ciao” non è un intero!*
- *http://127.0.0.1:8000/welcome_path/Mario/2.3/*
  - *KO: “2.3” non è un intero!*

In questo caso, vi è un vero e proprio type enforcement. Tipi di dato non corrispondenti previene il match tra l’url inserito e le regole di generazione descritte in urlpatterns