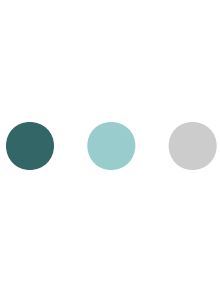


Django

Bista-geruza (*view layer*)

Bista-geruza (view layer)

- Djangoren bista-geruzan:
  - Bideratze edo *routing* sistema bat definitzen da. Eskaeraren URLa aztertu (espresio erregularren bitartez), eta erabakitzen da zein bistak prozesatuko duen eskaera hori.
  - Bistak Python funtzioak dira, HTTP eskaera bat jaso eta erantzuna sortzen dutenak. Erantzun hori txantiloi baten bitartez aurkeztuko da, txantiloi-geruzan zehazten den eran.
- Bista-geruzaren atal bakoitzari (bideratze-atala eta bisten atala) fitxategi bana dagokio:
  - Bideratze-atala (*dispatcher*): **`urls.py`**
  - Bisten atala: **`views.py`**

Fitxategien antolaketa

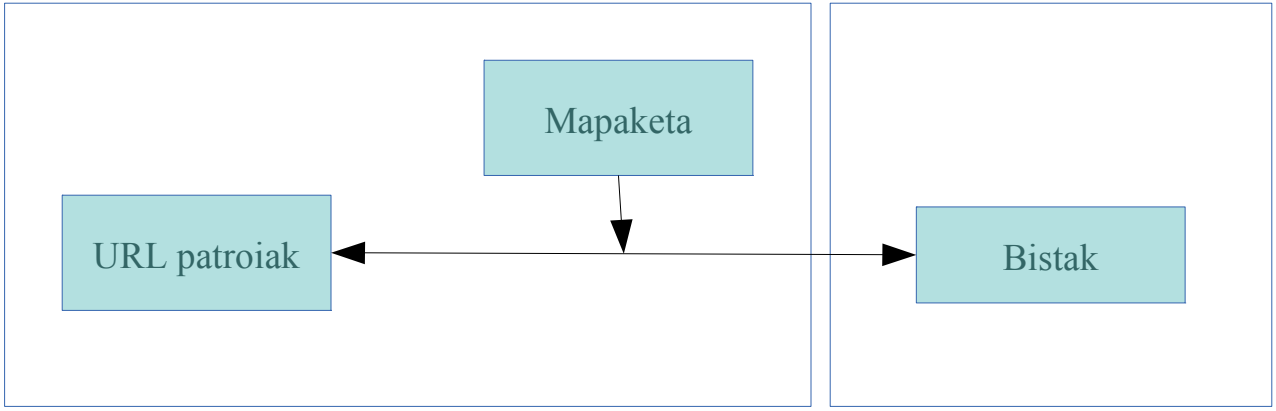
```
nireproiektua/
    manage.py
    nireproiektua/
        __init__.py
        settings.py
        urls.py
        wsgi.py

    aplikazioa/
        __init__.py
        models.py
        tests.py
        views.py
```

```
ROOT_URLCONF = 'nireproiektua.urls'
```

```
# URLconfig modulua
urlpatterns = patterns('',
    url(r'^admin/', include(admin.site.urls)),
)
```

URL abiarazlea (*URL dispatcher*)

- Django estiloan, URLak osatzean inplementazioari lotutako ezaugarriak desagertu egiten dira (fitxategi-luzapenak eta bestelako xehetasun itsusgarriak).
- Ikusi dugunez, aplikazio bateko URLak diseinatzeko *URLconf* modulu bat sortzen da.



URLconf: `urls.py` `views.py`

Erabiltzaile-eskaeren prozesaketa

1. Djangok erabakitzen du zein den erabili beharreko *URLconf* modulua. Besterik ezean, *ROOT_URLCONF* ezarpenean adierazitakoa da.
2. Djangok URLconf modulu hori kargatzen du, eta aztertzen du `urlpatterns` aldagaia. Aldagai hau lista bat da.
3. Djangok patroi-parekaketa prozesu bat jartzen du martxan (espresio erregularretan oinarritzen da prozesu hau): banan-banan aztertzen ditu patroiak, ordena errespetatuz, eta URLarekin parekatzen den lehenengoa topatzen duenean gelditzen da.
4. Patroi horri egokitutako bista inportatu eta bista-deia burutzen du. Bistak *HttpRequest* bat jasotzen du lehenengo argumentu gisa, eta horrez gain, espresio erregularrak harrapatu dituen gainontzeko balioak ere argumentu gisa jasotzen ditu.
5. Patroi-parekaketarik gertatu ez bada, edo salbuespenen bat gertatu bada, erroreak maneiatzeko bistari deitzen dio Djangok.

URLconf adibidea

```
from django.conf.urls import patterns, url, include

urlpatterns = patterns('',
    (r'^articles/2003/$', 'news.views.special_case_2003'),
    (r'^articles/(\d{4})/$', 'news.views.year_archive'),
    (r'^articles/(\d{4})/(\d{2})/$', 'news.views.month_archive'),
    (r'^articles/(\d{4})/(\d{2})/(\d+)/$', 'news.views.article_detail'),
)
```

URLconf Erabilera-oharrak

- Hasierako 'r' karakterea gomendagarria da: “raw string”, ihes-karaktererik ez.
- `^articles` jartzen da, ez `^/articles`: beti dago-eta hasieran “/” bat.
- Parentesiek URLko balioak kapturatzeko balio dute.
- URLconf-ek **ez du parekatzen domeinu-izena**ren kontra.
  - Adibidez: `http://www.example.com/myapp/` eskaeran, `myapp/` bilatzen du.
- URLconf-ek **ez du parekatzen GET edo POST parametroen kontra**.
  - Adibidez: `http://www.example.com/myapp/?page=3` eskaeran, `myapp/` bilatzen du.

`patterns` listaren egitura

- Lehenengo elementua: bisten aurrizkia. Adibidean: `''`
- Gainontzeko elementuak tuplak dira:
  - Tupla horietan derrigorrezko osagaiak bi dira: patroia eta bista-deia.
  - `url()` funtzioa ere erabil daiteke

```
Urlpatterns = patterns('news.views',
      url(r'^articles/2003/$', 'special_case_2003', name='2003_view'),
      ...
)
```

Espresio erregularrak

- Karaktere berezi ohikoenak:
  - '.': Edozein karaktere.
  - '^': String-aren hasiera.
  - '$': String-aren bukaera.
  - '*': n>=0 aldiz.
  - '+': n>=1 aldiz.
  - {m}: m aldiz.
  - '\': Ihes-karakterea.
  - []: karaktere-multzoa adierazteko.
  - '|': edo.
  - **(...): Taldea (*group*)** adierazteko eta mugatzeko. Parentesi barruko edozein espresio erregular parekatzen du. Talde bateko **edukiak kapturatu** eta berreskuratu egin daitezke parekaketa gertatu ostean.
  - **(?P<*izena*>...)**: **Talde izenduna** adierazteko. Taldeak parekatzen duen azpi-stringa *izena* erabiliz erreferentzia daiteke.

Espresio erregularrak (II)

- \d: Digitu hamartarra.
- \D: Digitua ez den beste edozein karaktere.
- \s: Zuriune karakterea (*whitespace)*.
- \S: Zuriunea ez den beste edozein karaktere.
- \w: Edozein karaktere alfanumeriko eta *underscore*a
- \W: Edozein karaktere ez-alfanumeriko.

- Gehiago sakontzeko: https://docs.python.org/2/library/re.html

Balioen kapturak patroietan

- Ikusi dugun adibidean URLko balioak kapturatzen dira, baina ez dira izendatzen eta bistetara modu posizionalean pasatzen dira.
  - Adibidez, `/articles/2005/03/` URLera egindako eskaerak honako bista-deia eragingo luke:
    - `news.views.month_archive(request, '2005', '03')`
- Posible da, ordea, kapturak izendatzea, eta gako-hitz argumentu gisa (*keyword argument*) pasatzea bistetara. Horretarako:
  - `(?P<name>pattern)`
  - Adibidez:

```
urlpatterns = patterns('',
    …,
    (r'^articles/(?P<year>\d{4})/(?P<month>\d{2})/$', 'news.views.month_archive'),
    ...,
)
```

  - Honakoan, `/articles/2005/03/` URLera egindako eskaerak honako bista-deia eragingo luke:
    - news.views.month_archive(request, year='2005', month='03')
- Kapturatutako balioak beti “string” gisa pasatzen zaizkio bistari.

Bisten aurrizkiak

- Aurreko adibidea horrela berridatz (eta laburtu) daiteke:

```
urlpatterns = patterns('news.views',
    (r'^articles/(\d{4})/$', 'year_archive'),
    (r'^articles/(\d{4})/(\d{2})/$', 'month_archive'),
    (r'^articles/(\d{4})/(\d{2})/(\d+)/$', 'article_detail'),
)
```

- Bi lista kateatuz, bi aurrizki desberdin adieraztea posiblea da:

```
urlpatterns = patterns('news.views',
    ...,
)
urlpatterns += patterns('old.news.views',
    ...,
)
```

Bista-deiak *URLconf*en *include*

- *include*-ren bitartez, beste nonbait definitutako *URLconf* bat txerta daiteke uneko *URLconf*ean.
  - Uneko *URLconf*a gurasoa izango da txertatuarekiko. Nolabait, aplikazio gurasoaren eta aplikazio instantziaren arteko bereizketa egiten da horrela.
  - Kapturatutako balioak pasa daitezke *URLconf* gurasotik *URLconf* txertatura.
  - Adibidez::

```
# app/urls.py
urlpatterns = patterns('',
    (r'^(?P<username>\w+)/blog/', include('blog.urls')),
)

# blog/urls.py
urlpatterns = patterns('',
    (r'^$', 'blog.index'),
    (r'^archive/$', 'blog.archive'),
)
```

URLconf izen-espazioak (*namespace)*

- Aplikazio nagusi baten instantzia anitz aktibatzen direnean, instantzia horien artean bereizteko gai izatea beharrezkoa da.
- Horretarako *namespace* kontzeptua erabiltzen da Djangon.
- Adibidez, *include* bat egiten denean, txertatutako *URLconf* horri instantziako izen-espazio bat eslei dakioke:

```
(r'^help/', include('apps.help.urls', namespace='help', app_name='guraso')),
```

URL patroi izendunak (*named URL patterns*)

- Ohikoa da URL patroi bat baino gehiagok bista-dei berdinak egitea.
  - Horrek arazoak sor ditzake alderantzizko bidea egin nahi denean: bistetatik URLetara (ikusiko da aurrerago).
  - Konponbidea: URL patroia izendatzea.
  - Adibidez:

```
urlpatterns = patterns('',
    url(r'^archive/(\d{4})/$', archive, name="full-archive"),
    url(r'^archive-summary/(\d{4})/$', archive, {'summary': True}, name="arch-summary"),
)
```

Bista-deiak
Stringen ordez, funtzioak

- Patroietan, posible da funtzioen izenak string gisa jarri ordez, zuzenean funtzio-objektuak jartzea.
- Adibidez:

```
from mysite import views

urlpatterns = patterns('',
    (r'^archive/$', views.archive),
    (r'^about/$', views.about),
    (r'^contact/$', views.contact),
)
```

Klasetan oinarritutako bistei eginiko deiak

- Klasetan oinarritutako bistek (*class-based views*) zeregin monotono batzuk orokortzeko eta ez errepikatzeko aukera ematen dute.
- Bistak definitzean, bista-klaseak sortzen badira, *URLconf*etik bista horiek erreferentziatzen dira. Horrela:

```
# nire_app/views.py
from django.views.generic import TemplateView

class AboutView(TemplateView):
    template_name = "about.html"
```

```
# urls.py
...
from nire_app.views import AboutView

urlpatterns = patterns('',
    (r'^about/', AboutView.as_view()),
)
```

URLen lanketarako zenbait funtzio

- `django.core.urlresolvers` modulukoak dira
  - **`reverse()`**: bista jakin batetik, URLra
    - `reverse(viewname[, urlconf=None, args=None, kwargs=None, current_app=None])`
    - Adibidez:

```
>>> reverse('admin:app_list', kwargs={'app_label': 'auth'})
    '/admin/auth/'
```

  - **`reverse_lazy()`**: reverse-ren bertsioa, *lazy* moduluan ebaluatzen dena.
    - `reverse_lazy(viewname[, urlconf=None, args=None, kwargs=None, current_app=None])`
  - **`resolve()`**: esaten du zein bista dagokion path (URL) jakin bati.
    - `resolve(path, urlconf=None)`

```
# Resolve a URL
match = resolve('/some/path/')
# Print the URL pattern that matches the URL
print match.url_name
```

Bisten atala

- Bista-funtzioak, edo bistak (*views*), Python funtzioak dira, Web eskaera hartu eta Web erantzuna itzultzen dutenak.
  - Erantzuna: HTML dokumentua, birbideratze bat, errore-mezua, multimedia dokumentua...
  - Edonon jar daitezke, baina ohitura ona da aplikazioaren *views.py* fitxategian definitzea.
  - Eskaera jaso eta erantzuna sortu arteko logika guztia bistan idazten da.
  - HTTP eskaerak prozesatzean gerta daitezkeen erroreak ere, bistan lantzen dira.
  - Bista bat URL jakin batekin lotzeko, URLconf bat definitu behar da, ikusi dugunez.

```
from django.http import HttpResponse
import datetime

def uneko_datadenbora(request):
    orain = datetime.datetime.now()
    html = "<html><body>Orain: %s.</body></html>" % orain
    return HttpResponse(html)
```

HttpRequest eta *HttpResponse* objektuak

- Objektu hauek dira edozein bistak maneiatu behar dituenak.
- Objektu hauek beren atributu eta metodoak dituzte. CGI interfazearekin zerikusi estua dute. Adibidez:
  - `HttpRequest:`
    - `HttpRequest.path, HttpRequest.method, HttpRequest.GET, HttpRequest.POST, HttpRequest.COOKIES, HttpRequest.FILES, HttpRequest.META, HttpRequest.user, HttpRequest.session, HttpRequest.urlconf...`
  - `UploadedFile:`
    - `UploadedFile.name, UploadedFile.size`
  - `QueryDict:`
    - GET eta POST atributuak QueryDict klasearen instantziak dira. Klase honek hiztegi (array asoziatibo) baten gisako erabilera du.
  - `HttpResponse`:
    - Argumentu gisa har ditzake stringak, iteratzaileak, goiburuko eremuak (hiztegi gisa)
    - `HttpResponse.content, HttpResponse.status_code, HttpResponse.__setitem__(header, value), HttpResponse.__getitem__(header), HttpResponse.write(content)`
    - `HttpResponseRedirect` azpiklasea
- http://teknikariak.informatika.ehu.es/Django/ref/request-response.html

Shortcut funtzioak

- Funtzio hauek `django.shortcuts` paketean daude, eta hainbat zereginetan laguntza eskaintzen dute. Horien erabileraren bidez, laburtu egiten da idatzi beharreko kodea.
- Testuinguru honetan erabilgarriak:
  - **`render`** eta **`render_to_response`**: testuinguru bat emanda, txantiloi baten bidez renderizatzeko
  - **`redirect`**: birbideraketak egiteko.
  - **`get_object_or_404`**: eredu-maneiatzaile bati dei egitean, `DoesNotExist` salbuespenaren ordez `Http404` sortzen da.

Osatzeko...

- http://teknikariak.informatika.ehu.es/Django/topics/http/urls.html
- http://www.djangobook.com/en/2.0/chapter03.html
- http://www.djangobook.com/en/2.0/chapter08.html