Type Check Django App - Euro Python 2021

Types at runtime

```
type(("127.0.0.1", 8000))

# output
tuple
```

```
from django.contrib.auth.models import User


type(User.objects.filter(
email='foo@example.com'))

# output
django.db.models.query.QuerySet
```

STATIC CHECKER TYPE

PYTHON FILE

```
#filename.py
addr = "127.0.0.1"
port = 8000

reveal_type((addr, port))
```

```
$mypy filename.py
note: Revealed type is
'Tuple[builtins.str, builtins.int]'
```

```
#filename.py
from django.contrib.auth.models import User

reveal_type(User.objects.filter(
email='foo@example.com'))
```

```
$mypy filename.py
note: Revealed type is
'django.contrib.auth.models.UserManager
[django.contrib.auth.models.User]'

$cat mypy.ini
...
plugins =
mypy_django_plugin.main,

[mypy.plugins.django-stubs]
django_settings_module = "yourapp.settings"
```

ANNOTATION SYNTAX

VARIABLE

```python
from datetime import date
lang: str = "Python"
year: date = date(1989, 2, 1)
```

FUNCTION

```
def sum(a: int, b: int) -> int:
    return a + b

class Person:
     def __init__(self, name: str, age: int,
                  is_alive: bool):
         self.name = name
         self.age = age
         self.is_alive = is_alive
```

ANNOTATING DJANGO CODE

VIEW

```python
from django.http import (HttpRequest, HttpResponse,
      HttpResponseNotFound)

def index(request: HttpRequest) -> HttpResponse:
    return HttpResponse("hello world!")

def view_404_0(request: HttpRequest) -> HttpResponse:
    return HttpResponseNotFound(
      'Page not found')
```

ALTERNATE

```
def view_404_1(request:
               HttpRequest) -> HttpResponseNotFound:
    return HttpResponseNotFound(
      'Page not found')

# bad - not precise and not useful
def view_404_2(request: HttpRequest) -> object:
  return HttpResponseNotFound(
      'Page not found')
```

```
HttpResponse.mro()
[django.http.response.HttpResponse,
 django.http.response.HttpResponseBase,
 object]


HttpResponseNotFound.mro()
[django.http.response.HttpResponseNotFound,
 django.http.response.HttpResponse,
 django.http.response.HttpResponseBase,
 object]
```

LSP

The LSP states
that in an **object-oriented program**,
substituting a superclass
object reference with an object
of any of its subclasses,
the program should not **break**.

SUMMARY

```
While designing objects and annotating return values
try adhering to LSP.
Mypy will complain when the return
value deviates from LSP
```

DJANGO MODELS

CREATE

```python
from django.db import models
from django.utils import timezone


class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField("date published")


def create_question(question_text: str) -> Question:
    qs = Question(question_text=question_text,
                  pub_date=timezone.now())
    qs.save()
    return qs
```

READ

```
def get_question(question_text: str) -> Question:
    return Question.objects.filter(
      question_text=question_text).first()
```

```
error: Incompatible return value type
(got "Optional[Any]", expected "Question")
```

OPTIONAL

```
from typing import Optional

def get_question(question_text: str) ->
  Optional[Question]:
    return Question.objects.filter(
      question_text=question_text).first()
```

MYPY CONFIG

```
# mypy.ini
strict_optional = False

def get_question(question_text: str) ->
  Question:
    return Question.objects.filter(
        question_text=question_text).first()
```

FILTER

EXAMPLE

```
In [8]: Question.objects.all()
Out[8]: <QuerySet [<Question: Question object (1)>,
            <Question: Question object (2)>]>

In [9]: Question.objects.filter()
Out[9]: <QuerySet [<Question: Question object (1)>,
                <Question: Question object (2)>]>
```

```
def filter_question(text: str) ->
  QuerySet[Question]:
    return Question.objects.filter(
      text__startswith=text)

def exclude_question(text: str) ->
  QuerySet[Question]:
    return Question.objects.exclude(
      text__startswith=text)
```

METHODS RETURNING QUERYSET

```
all, reverse, none, complex_filter,
union, order_by, distinct,
defer, only, using, extra,
select_related, select_for_update,
prefetch_related
```

AGGREGATE

```python
class Author(models.Model):
    name = models.CharField(max_length=100)
    age = models.IntegerField()

class Publisher(models.Model):
    name = models.CharField(max_length=300)

class Book(models.Model):
    name = models.CharField(max_length=300)
    pages = models.IntegerField()
    # use integer field in production
    price = models.DecimalField(max_digits=10,
                                decimal_places=2)
    rating = models.FloatField()
    authors = models.ManyToManyField(Author)
```

EXAMPLE

```
def get_avg_price():
    return Book.objects.all().aggregate(
      avg_price=Avg("price"))

print(get_avg_price())
{'avg_price': Decimal('276.666666666667')}
```

ANNOTATED CODE

```
from decimal import Decimal


def get_avg_price() -> dict[str, Decimal]:
    return Book.objects.all().aggregate(
      avg_price=Avg("price"))
```

ANNOTATE METHOD

COUNT BOOKS BY PUBSLISHER

```
def count_by_publisher():
    return Publisher.objects.annotate(
      num_books=Count("book").values(
        'name', 'num_books'))
```

RESULT

```
In [17]: [i for i in count_by_publisher()]
Out[17]: [{'name': 'Penguin', 'num_books': 2},
          {'name': 'vintage', 'num_books': 1}]
```

RETURNING QUERYSET

```
def count_by_publisher():
    return Publisher.objects.annotate(
      num_books=Count("book"))


def print_pub(num_books=0):
    if num_books > 0:
        res = count_by_publisher().filter(
          num_books__gt=num_books)
    else:
        res = count_by_publisher()
    for item in res:
        print(item.name, item.num_books)
```

EXAMPLE

```
In [42]: print_pub()
Penguin 2
vintage 1
```

ANNOTATION

```
from typing import TypedDict
from collections.abc import Iterable

class PublishedBookCount(TypedDict):
    name: str
    num_books: int

def count_by_publisher() ->
  Iterable[PublishedBookCount]:
  ...
```

ERROR

```
# mypy output
scratch.py:46: error: Incompatible return value
    type (got "QuerySet[Any]", expected
"PublishedBookCount")
        return Publisher.objects.annotate(
          num_books=Count("book"))
                 ^
scratch.py:51: error:
      "PublishedBookCount" has no attribute "filter"
       res = count_by_publisher().filter(
         num_books__gt=num_books)
```

QUERYSET

```
def count_by_publisher() -> QuerySet[Publisher]:
    ...

def print_pub(num_books: int=0) -> None:
    ...
    for item in res:
        print(item.name, item.num_books)
```

ERROR

```
# mypy output

scratch.py:55: error: "Publisher" has
      no attribute "num_books"
      print(item.name, item.num_books)
```

BUG

https://github.com/typeddjango/django-stubs/pull/398

SUGGESTION

TYPE CHECKING

```
from typing import TYPE_CHECKING

if TYPE_CHECKING:
    class TypedPublisher(Publisher):
        num_books = models.IntegerField()

        class meta:
            abstract = True


def count_by_publisher() -> QuerySet[TypedPublisher]:
    return Publisher.objects.annotate(
         num_books=Count("book"))
```

TOOLS

PYANNOTATE

- Automatically insert type-hints into the code
- pytest-annotate infer type from test cases
- Project Page

SOURCE CODE

```
from django.http import (HttpResponse,
    HttpResponseNotFound)

# Create your views here.
# annotate the return value
def index(request):
    return HttpResponse("hello world!")

def view_404_0(request):
    return HttpResponseNotFound(
      'Page not found')
```

TEST CODE

```
from polls.views import *
from django.test import RequestFactory

def test_index():
    request_factory = RequestFactory()
    request = request_factory.post('/index')
    index(request)


def test_view_404_0():
    request_factory = RequestFactory()
    request = request_factory.post('/404')
    view_404_0(request)
```

RUN TEST COMMAND

```
$DJANGO_SETTINGS_MODULE="mysite.settings"
PYTHONPATH='.' poetry run pytest
-sv polls/tests.py
--annotate-output=./annotations.json
```

PLUGIN OUTPUT

```
$cat annotations.json
[...
    {
        "path": "polls/views.py",
        "line": 7,
        "func_name": "index",
        "type_comments": [
            "(django.core.handlers.wsgi.WSGIRequest) ->
          django.http.response.HttpResponse"
        ],
        "samples": 1
    },
    {
        "path": "polls/views.py",
        "line": 10,
```

APPLY THE CHANGES

```
$poetry run pyannotate --type-info
./annotations.json
-w polls/views.py --py3
```

AUTOMODIFIED FILE

```
from django.http import HttpResponse, HttpResponseNotFound
from django.core.handlers.wsgi import WSGIRequest
from django.http.response import HttpResponse
from django.http.response import HttpResponseNotFound

def index(request: WSGIRequest) -> HttpResponse:
    return HttpResponse("hello world!")

def view_404_0(request: WSGIRequest) ->
  HttpResponseNotFound:
    return HttpResponseNotFound(
      'Page not found')
```

LEARNING RESOURCE

PYTHON TYPING KOANS

https://github.com/kracekumar/python-typing-koans

THANK YOU!

@kracetheking
GitHub - Kracekumar