

Django Views

Django Views

Контроллеры в Django - это обычные функции, которые:

- принимают объект `django.http.HttpRequest` первым параметром
- возвращают объект `django.http.HttpResponse`

Django Views

```
# /blog/post_text/?id=123
def post_text(request):
    try:
        id = request.GET.get('id')
        obj = Post.objects.get(pk=id)
    except Post.DoesNotExist:
        raise Http404
    return HttpResponse(obj.text,
                        content_type='text/plain')
```

Захват параметров из URL

```
# blog/urls.py
```

```
# /category/123/
```

```
path_re('^category/(\d+)/$', 'category_view')
```

```
# /123/
```

```
path_re('^(?P<pk>\d+)/$', 'post_detail')
```

```
# /123/
```

```
path('<int:pk>/', 'post_detail')
```

Захват параметров из URL (2)

```
# blog/views.py
```

```
def category_view(request, pk):  
    # Вывести все посты
```

```
def post_detail(request, pk=None):  
    # Вывести страницу поста
```

```
def category_view(request, *args, **kwargs):  
    pk = args[0]  
    pk = kwargs['pk']
```

HttpRequest и
HttpResponse

HttpRequest

- `request.method` - метод запроса
- `request.GET` - словарь с GET параметрами
- `request.POST` - словарь с POST параметрами
- `request.COOKIES` - словарь с Cookie
- `request.FILES` - загруженные файлы
- `request.META` - CGI-like переменные
- `request.session` - словарь-сессия (*)
- `request.user` - текущий пользователь (*)

HttpResponse

```
from django.http import HttpResponse
# создание ответа
response = HttpResponse("<html>Hello world</html>")
# установка заголовков
response['Age'] = 120
# установка всех параметров
response = HttpResponse(
    content = '<html><h1>Ничего</h1></html>',
    content_type = 'text/html',
    status = 404,
)
```


Специальные типы ответов

```
from django.http import HttpResponseRedirect, \
    HttpResponseNotFound, HttpResponseForbidden, \
    HttpResponsePermanentRedirect

redirect = HttpResponseRedirect("/") # 302
redirect = HttpResponsePermanentRedirect("/") # 301
response = HttpResponseNotFound() # 404
response = HttpResponseForbidden() # 403
```

Получение GET и POST параметров

```
order = request.GET['sort'] # опасно!  
if order == 'rating':  
    queryset = queryset.order_by('rating')  
page = request.GET.get('page', 1)  
try:  
    page = int(page)  
except ValueError:  
    return HttpResponseRedirect()
```

GET и POST - объекты QueryDict

```
/path/?id=3&id=4&id=5
```

Получение множественных значений

```
id = request.GET.get('id')      # id is 5  
id = request.GET.getlist('id') # id is [3,4,5]
```

Сериализация

```
qs = request.GET.urlencode()  
# qs is 'id=3&id=4&id=5'
```

Получение и установка HTTP заголовков

```
user_agent = request.META.get('HTTP_USER_AGENT')
user_ip = request.META.get('HTTP_X_REAL_IP')
if user_ip is None:
    user_ip = request.META.get('REMOTE_ADDR')

response = HttpResponse(my_data,
    content_type='application/vnd.ms-excel')
response['Content-Disposition'] = \
    'attachment; filename="foo.xls"'
```

Получение и установка Cookie

```
response = HttpResponse(html)
response.set_cookie('visited', '1')
# Set-Cookie: visited=1
```

```
is_visited = request.COOKIES.get('visited')
# Cookie: id=a3fWa; visited=1; test=aefw3
```

Декораторы

Декораторы в Python

Декоратор – функция, преобразующая одну функцию в другую.

```
def double_it(func):  
    def tmp(*args):  
        return func(*args) * 2  
    return tmp
```

```
@double_it  
def mult(a, b):  
    return a*b
```

```
# Добавление декоратора аналогично вызову  
mult = double_it(mult)
```

Декораторы в Django

```
from django.views.decorators.http import require_POST
@require_POST
def like(request):
    pass
```

- `@require_GET` – только GET запросы
- `@require_POST` – только POST запросы
- `@login_required(login_url='/login/')`
- `@csrf_exempt` – отключить проверку CSRF

Class-based Views

TemplateView

```
# views.py
from django.views.generic.base import TemplateView

class HomePageView(TemplateView):
    template_name = "home.html"

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context["additional_context"] = "some data"
        return context

# urls.py
urlpatterns = [
    path('', HomePageView.as_view(), name='home'),
]
```

Class-based Views

- `TemplateView` – рендеринг шаблона
- `RedirectView` – редирект
- `DetailView` - просмотр одной сущности
- `ListView` – просмотр листинга
- `FormView` – работа с формами
- ...

[Больше классов в документации](#)

Шаблонизация

Неправильный подход

```
def header():  
    return '<html><head>...</head><body>'  
  
def footer():  
    return '</body></html>'  
  
def page1(data):  
    return header() + \  
        '<h1>' + data['title'] + '</h1>' + \  
        '<p>' + data['text'] + '</p>' + \  
        footer()
```

Правильный подход

Необходимо разделить данные (**контекст**) от представления (**шаблона**). Для этого используются **шаблонизаторы**.

- + Разделение работы web-мастера и программиста
- + Повторное использование HTML кода
- + Более чистый python код

Синтаксис шаблонов

```
<!-- templates/blog/post_detail.html -->
<html>
  <head>...</head>
  <body>
    <h1>{{ post.title|truncatechars:80 }}</h1>
    <p>{{ post.text }}</p>
    {% for comment in comments %}
      {% include "blog/comment.html" %}
    {% endfor %}
  </body>
</html>
```

Вызов шаблонизатора

```
# project/blog/views.py
from django.shortcuts import render

return render(request, 'blog/post_detail.html', {
    'post': post,
    'comments': comments,
})
```


Возможности шаблонизатора

- `{% for item in list %}{% endfor %}` - итерация по списку
- `{% if var %}{% endif %}` - условное отображение
- `{% include "tpl.html" %}` - включение подшаблона
- `{{ var }}` - вывод переменной
- `{{ var|truncatechars:9 }}` - применение фильтров
- `{# comment #}`, `{% comment %}{% endcomment %}` - комментарии

Доступ к свойствам и методам

Через точку можно получить свойство, метод, ключ либо индекс объекта:

```
{{ object.content }}  
{{ name.strip }}  
{{ info.avatar }}  
{{ post_list.0 }}
```

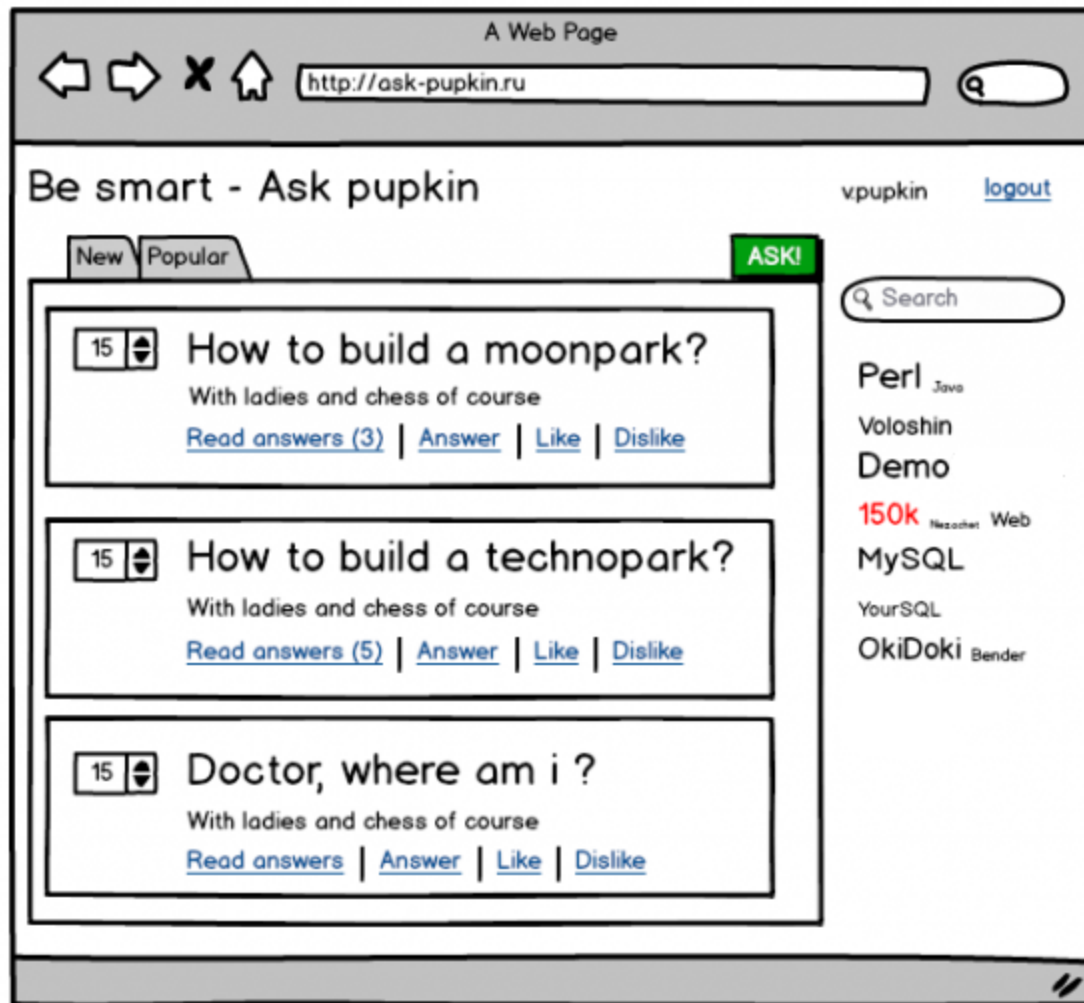
Передавать параметры методам запрещено:

```
{{ post_list.order_by('id') }} <!-- ошибка -->  
{{ post_list.delete }}
```

Особенности шаблонизатора

- Шаблоны автоматически экранируют HTML сущности
- Шаблонизатор можно расширять своими фильтрами и тэгами

Наследование шаблонов



Базовый шаблон base.html

```
<!DOCTYPE HTML>
<html>
<head>
    <title>{% block title %}Q&A{% endblock %}</title>
    {% block extrahead %}{% endblock %}
</head>
<body>
    <h1>Вопросы и ответы</h1>
    {% block content %}{% endblock %}
</body>
</html>
```

Шаблон главной страницы

```
{% extends "base.html" %}
{% block title %}
    {{ block.super }} – главная
{% endblock %}
{% block content %}
    {% for obj in post_list %}
        <div class="question">
            <a href="{{ obj.build_url }}">{{ obj }}</a>
            {{ obj.created_date|date:"d.m.Y" }}
        </div>
    {% endfor %}
{% endblock %}
```

Context processors

Context processors

Context processors - это функции, которые вызываются перед отрисовкой шаблона и могут добавить данных в контекст.

Настройка `context_processors` в `TEMPLATES`:

- `django.template.context_processors.request` (request)
- `django.template.context_processors.csrf` (csrf_token)
- `django.template.context_processors.static` (STATIC_URL)
- `django.contrib.auth.context_processors.auth` (user, perms)