МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

Урок №3

Сервер приложений

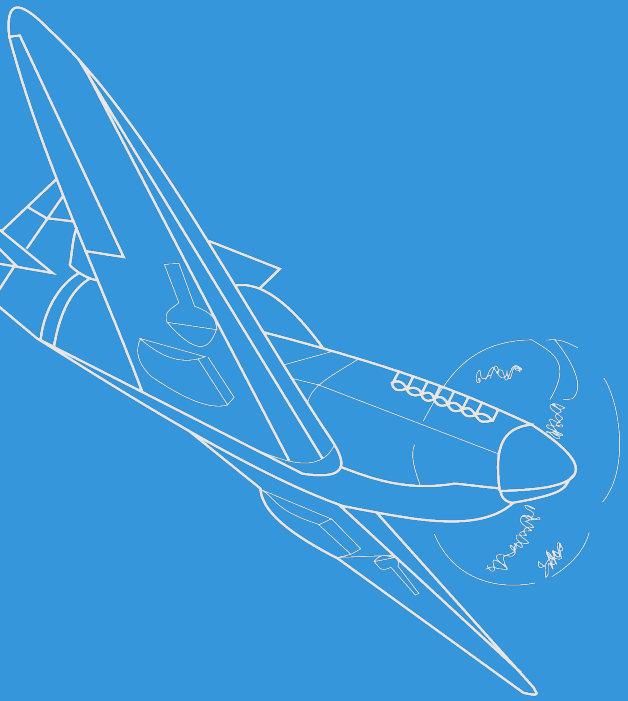
на котором расскажут про MVC, Django, маршрутизацию, шаблонизацию, HttpRequest, HttpResponse.

28 сентября 2020 года

Антон Кухтичев

Содержание занятия

1. Сервер приложений
2. Модель MVC
3. Django приложения
4. Django views
5. Шаблонизация



Сервер приложений

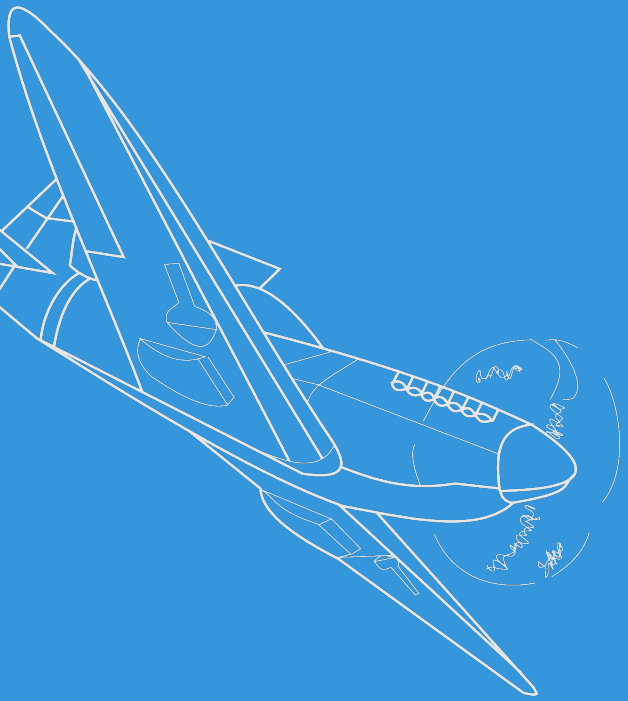
Основные виды запросов, задачи.

Основные типы запросов

- Запросы статических документов;
- Запросы динамических документов;
- Отправка данных форм;
- AJAX-запросы;
- Запросы к API сайта;
- Персистентные соединения.

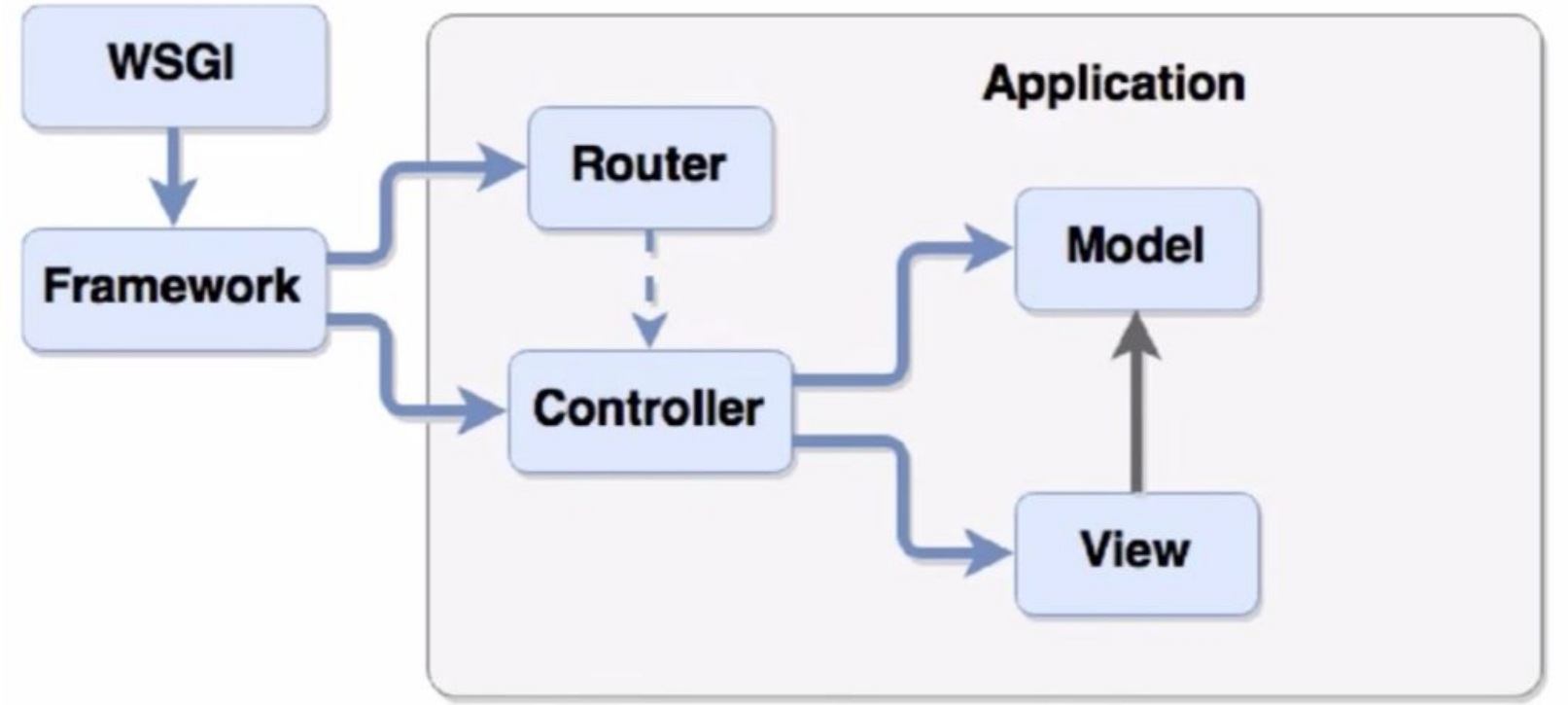
Основные задачи

- Маршрутизация URL;
- Парсинг заголовков и параметров запроса;
- Хранение состояния (сессии) пользователя;
- Выполнение бизнес-логики;
- Работа с базами данными;
- Генерация HTML-страницы или JSON-ответа.



MVC

Model-View-Controller



Роли компонентов MVC

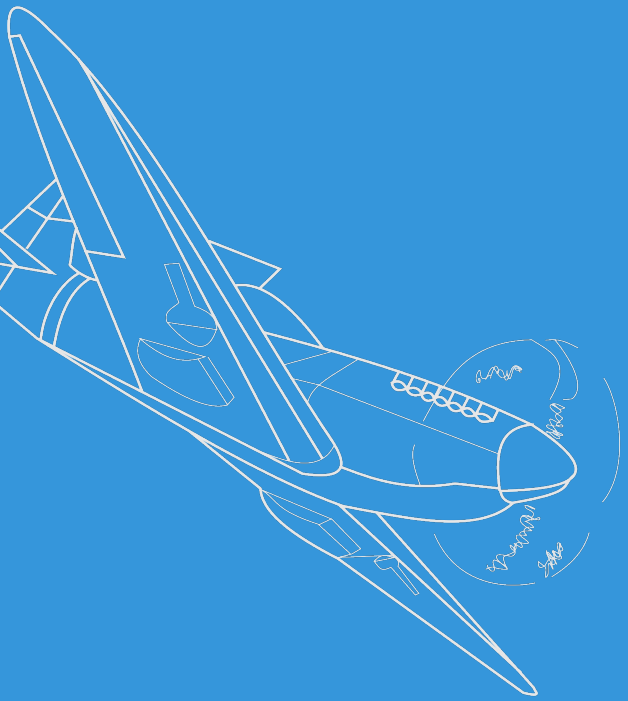
- `Router` — выбор конкретного controller по URL;
- `Model` — реализация бизнес-логики приложения;
- `Controller` — работа с HTTP, связь controller и view;
- `View` — генерация HTML или другого представления.

Фреймворк

Плюсы фреймворков

- Готовая архитектура;
- Повторное использование кода;
- Экономия ресурсов;
- Участие в Open Source;
- Проще найти программистов;
- Проще обучать программистов.

Django



Установка

```
(venv) pip install Django==2.0
(venv) pip freeze > requirements.txt
```

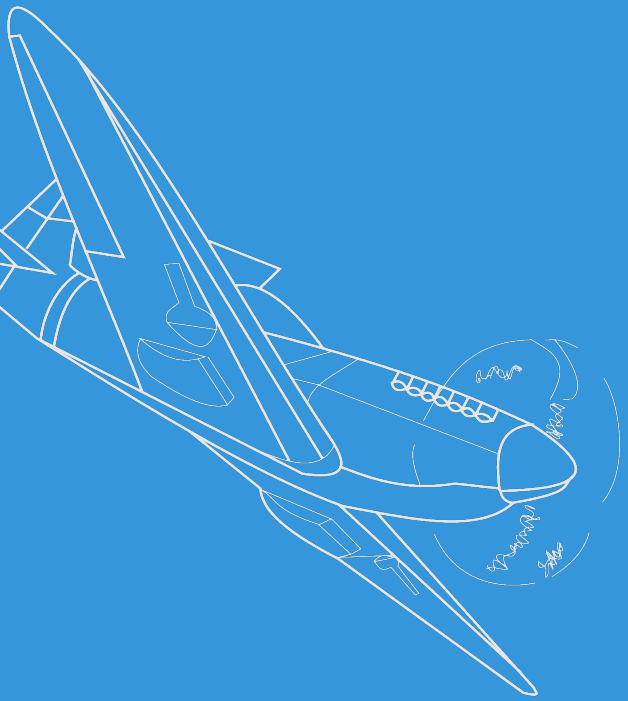
Соглашение о именовании

MVC	**Django**
Model	Model
Router	urls.py
Controller	views
View	templates

Структура проекта

`django-admin startproject project` — создание проекта.

```
project
├── crm
│   ├── apps.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
├── manage.py
└── project
    ├── asgi.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```



Конфигурация Django

Конфиг — просто python модуль


```
# project/project/settings.py
ROOT_URLCONF = 'project.urls'
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3')
    }
}
TEMPLATE_DIR = os.path.join(BASE_DIR, 'templates')
```

Пути в конфиге

Проблемы:

- Проект может быть развернут в любой директории;
- Несколько копий проекта на одном сервере.

Решения:

- Абсолютные пути в каждом конфиге
- Переменные окружения, `${PROJECT_PATH}`
- Относительные пути;

local_settings.py


```
# в конце project/settings.py
try:
    from .local_settings import *
except ImportError:
    pass
```

Маршрутизация в проекте

- Django начинает поиск с файла `ROOT_URLCONF`;
- Загрузив файл, Django использует переменную `urlspattern`;
- Проходит по всем паттернам до первого совпадения;
- Если не найдено — 404.

Маршрутизация в проекте


```
# project/project/urls.py
from django.urls import path, re_path
from django.contrib import admin

urlpatterns = [
    path('', index, name='index'),
    re_path(r'^$', 'chats.views.index',
                   name='index'),
    path('chats/', include('chats.urls')),
    path('admin/', admin.site.urls),
]
```

Маршрутизация в приложении

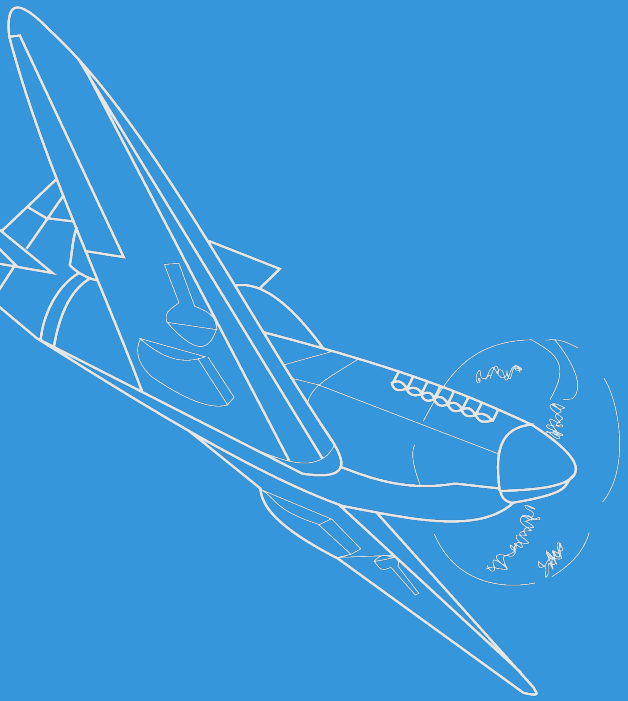

```
# messenger/chats/urls.py
from chats.views import chat_list
urlpatterns = [
    path('', chat_list, name='chat_list'),
    path('category/<int:pk>/',
         'chat_category',
         name='chat_category'),
    path('<chat_id>/', 'chat_detail', name='chat_detail'),
]
```

Особенности маршрутизации в Django

- Слеш (/) в начале роутеров не указывается;
- Можно указывать как имя, так и саму view-функцию;
- Роуты описываются с помощью регулярных выражений;
- Можно и нужно разносить роуты по приложениям;
- Можно и нужно создавать именованные роуты;
- Одно действие — один роут — один контроллер;

Reverse routing


```
from django.core.urlresolvers import reverse
reverse('index')
reverse('blog_category', args=(10, ))
reverse('blog_detail', kwargs={'pk': 2})
# в шаблонах
{% url 'blog_category' category_id %}
```

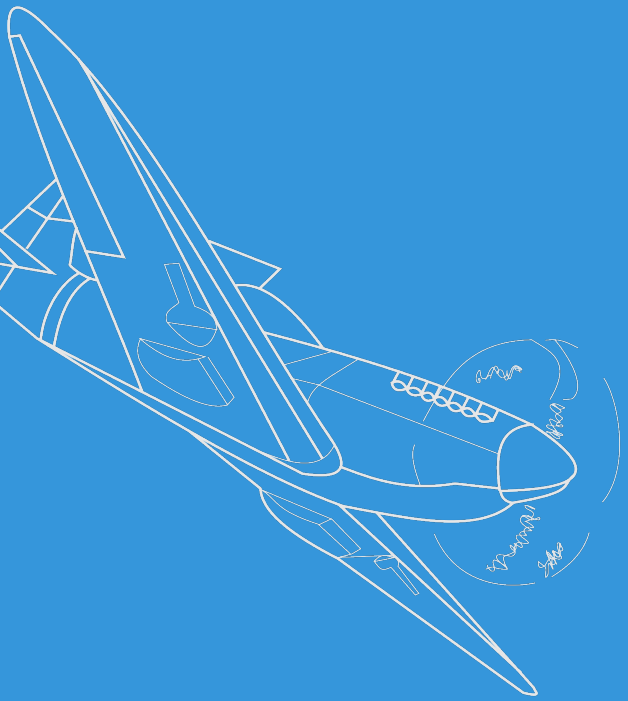


Django-приложения

Django-приложения

Приложения — способ распространения кода в Django-инфраструктуре. В случае, если вы не планируете публиковать ваш код, приложения - это просто способо логической организации кода.

`./manage.py startapp crm` — создание нового приложения с именем `crm`. Нужно вызывать из директории проекта.



Django views

Django views

Контроллеры в Django — это обычные функции, которые:

- Принимают объект `django.http.HttpRequest` первым параметром;
- Возвращает объект `django.http.HttpResponse`.

Django views


```
# project.blog.views
# запрос вида /blog/?id=2
def blog_detail(request):
    try:
        blog_id = request.GET.get('blog_id')
        blog = Blog.objects.get(id=blog_id)
    except Blog.DoesNotExist:
        raise Http404
    return HttpResponse(blog.description,
           content_type='text/plain')
```

Захват параметров из URL


```
# blog/urls.py

url(r'^category/(\d+)/$', 'category_view')
url(r'^(?P<pk>\d+)/$', 'post_detail')
```

Захват параметров из URL (2)


```
# blog/views.py
def category_view(request, pk=None):
    # вывести все посты

def post_details(request, pk):
    # вывести страницу поста

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

Django views

- `request.method` — метод запроса
- `request.GET` — словарь с GET параметрами
- `request.POST` — словарь с POST параметрами
- `request.COOKIES` — словарь с Cookies
- `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 HttpReponseRedirect, \
 HttpReponseNotFound, HttpResponseForbidden, \
 HttpResponsePermanentRedirect, JsonResponse

redirect = HttpReponseRedirect(“/”)             # 302
redirect = HttpResponsePermanentRedirect(“/”) # 301
response = HttpResponseNotFound()               # 404
response = HttpResponseForbidden()              # 403
response = JsonResponse({'foo': 'bar'})
```

GET и POST - объекты QueryDict

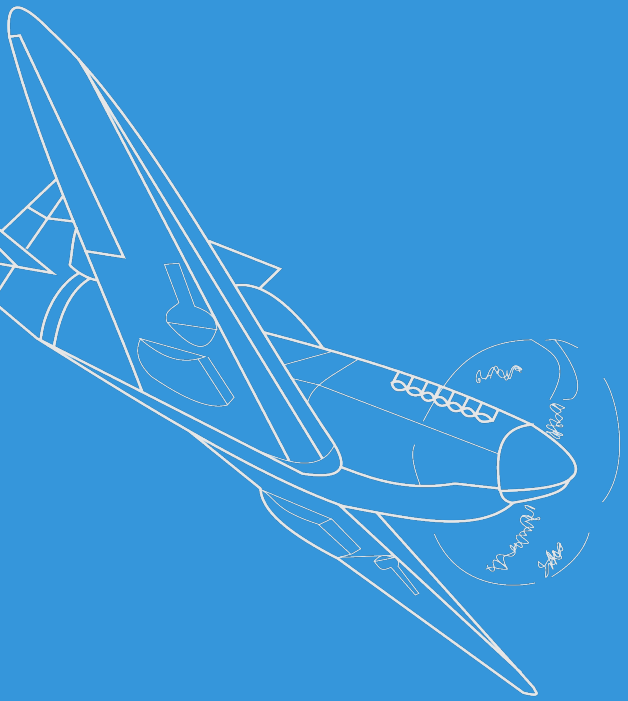

```
/path/?id=3&id=4&id=5
```

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'
```



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’] + ‘</p1>’ + \
           footer()
```

Правильный подход

Необходимо отделить 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_details.html →
<html>
    <head>...</head>
    <body>
        <h1>{{ post.title }}</h1>
        <p>{{ post.text }}</p>
        {% for comment in comments %}
            {% include “blog/comment.html” %}
        {% endfor %}
    </body>
</html>
```

Вызов шаблонизатора

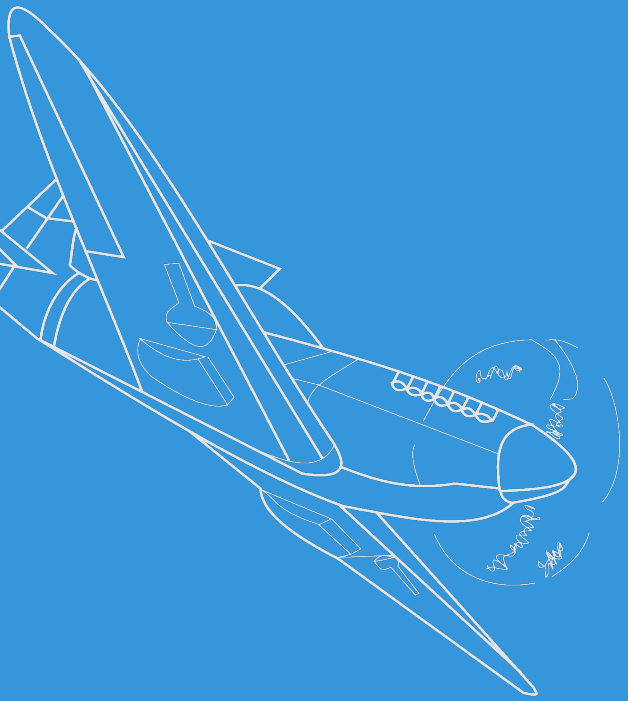

```
# project/blog/views.py
from django.shortcuts import render, render_to_response

return render_to_response('blog/post_details.html', {
        'post': post,
        'comments': comments,
})

return render(request, 'blog/post_details.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

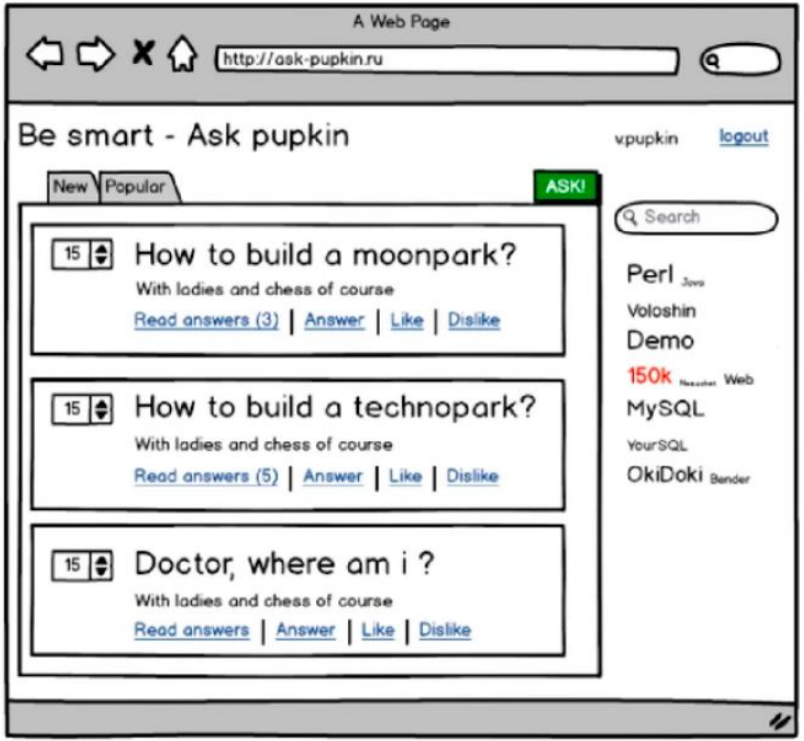
Особенности шаблонизатора

- Шаблона автоматически экранируют HTML сущности;
- Шаблонизатор можно расширять своими фильтрами и тэгами.



Наследование шаблонов

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 posts %}
        <div class=”question”>
            <a href=”{{ obj.build_url”>{{ obj }} </a>
        </div>
    {% endfor %}
{% endblock %}
```

Домашнее задание № 3

1. Создать и запустить Django-проект — 2 балла;
2. Реализовать «заглушки» для всех методов API, используя JsonResponse — 3 баллов:
   a. Профиль;
   b. Список продуктов;
   c. Страница категории;
   d. и т.д.
3. В конфиге nginx создать location, которые будет ходить на Django-приложение — 3 балла;
4. Обрабатывать только нужные методы (GET/POST) — 2 балла.

Рекомендуемая литература

[Документация Django]
[Code styleguide for Python PEP 8]
[Дока для pylint - проверка, что код соответствует PEP 8]

Для саморазвития (опционально)
[Чтобы не набирать двумя пальчиками]

Спасибо за внимание!

Антон Кухтичев

a.kukhtichev@mail.ru

@toshunster