Наведение порядка и подведение итогов

Деление проекта на приложения. Рекомендации по организации структуры. Модифицируем URL’ы. Несколько полезных инструментов. Подведение итогов.

[Наведем порядок](#_c713xnuw3xem)

[Деление на приложения (app)](#_68um2ek35me0)

[URLS - строки вместо импортов](#_6pf6en73xopz)

[Немного о моделях](#_r1vlvcr2v8np)

[Несколько полезных инструментов](#_h9ho0e1pteo)

[Декораторы](#_my6r5siafask)

[Собственные шаблонные фильтры](#_x2ky17fzozxk)

[Теория](#_92lnvkhx3bht)

[Практическая часть урока](#_gh6qjqz6p1kr)

[Подведем итоги](#_d0myqbz4noif)

[Домашнее задание](#_owpsqd9opgcf)

[Дополнительные материалы](#_y937sk8fclye)

[Используемая литература](#_ci4mlo7ijapz)

## 

## 

## 

Примечание: Данный урок преднамеренно сделан короче предыдущих, т.к. бОльшую его часть планируется посвятить разбору предыдущих ДЗ и ответам на накопившиеся вопросы по пройденному материалу.

# Наведем порядок

Я думаю, вы обратили внимание, насколько быстро увеличивается количество файлов в проекте. Найти нужную информацию становится все сложнее и сложнее, html’ки “свалены кучей”, во views.py функции-обработчики перемешиваются. Чтобы избежать подобного хаоса, надо постоянно развивать и поддерживать структуру проекта.

## Деление на приложения (app)

Приложения - это логическое деление проекта на части для удобства разработки, а также возможности последующего использования отдельных приложений в других проектах. Каждое приложение Django состоит из пакета Python, который следует некоторым соглашениям.

Разобьём на три приложения: MainApp, UserManagementApp и AdminApp.

MainApp - здесь будет весь код, ответственный за отображение страниц сайта.  
UserManagementApp - будет заниматься операциями с пользователями (регистрация, авторизация …)  
AdminApp - наша админка и всё, что с ней связано

Выполним в консоли:

|  |
| --- |
| python3 manage.py startapp UserManagementApp python3 manage.py startapp AdminApp |

Разбросаем функции представления  
MainApp/views.py

|  |
| --- |
| from django.shortcuts import render  from MainApp.models import Gem, Category  def main(request):  ...  def gems\_page(request):  ... |

UserManagementApp/views.py

|  |
| --- |
| from django.contrib import auth  from django.shortcuts import render, HttpResponseRedirect  from MainApp.forms import MyRegistrationForm  from django.http import Http404  def registration(request):  ...  def login(request):  ...  def logout(request):  ... |

**AdminApp/views.py**

|  |
| --- |
| from django.shortcuts import render, HttpResponseRedirect, get\_object\_or\_404  from django.contrib.auth.models import User  from django.http import Http404, JsonResponse  from django.template import loader  from django.template.context\_processors import csrf  from MainApp.forms import MyRegistrationForm, GemsForm  def admin\_page(request):  ...  def admin\_gems(request):  ...  def create\_gem(request):  ...  def delete\_user(request, user\_id):  ...  def create\_user(request, user\_id=None):  ...  def get\_user\_form(request, user\_id):  ... |

## URLS - строки вместо импортов

Теперь нам нужно изменить urls.py. Для организации url’ов при работе с несколькими приложениями можно использовать более удобный подход. Используем функцию patterns(), принимающую первым аргументом префикс.

Новый urls.py

|  |
| --- |
| from django.conf.urls import url, patterns  urlpatterns = patterns('MainApp.views',  url(r'^$', 'main'),  url(r'^gems/$', 'gems\_page')  )  urlpatterns += patterns('UserManagementApp.views',  url(r'^user/login/$', 'login'),  url(r'^user/logout/$', 'logout'),  url(r'^user/registration/$', 'registration'),  )  urlpatterns += patterns('AdminApp.views',  url(r'^admin/users/$', 'admin\_page'),  url(r'^admin/gems/$', 'admin\_page'),  url(r'^admin/delete/user/(\d+)$', 'delete\_user'),  url(r'^admin/get\_user\_form/(\d+)$', 'get\_user\_form'),  ) |

Преимущества:

* Не нужно делать много импортов;
* Пути к функциям-представлениям более наглядные;
* URL’ы сгруппированы по соответствующим приложениям, и при необходимости (например, при очень большом количестве url’ов) можно urls.py разбить на более мелкие части.

Еще может быть использован подяход с Include() для альтернативной организации url’ов, об этом [здесь](http://djbook.ru/ch08s02.html).

## Немного о моделях

Наше приложение содержит всего две модели, оставим их в основном приложении MainApps. Когда вы захотите создать пользователя с дополнительными полями CustomUser, его будет логично разместить в models.py приложения UserManagementApp.

# Несколько полезных инструментов

Прежде чем подвести итоги, рассмотрим несколько полезных инструментов из арсенала Django.

## Декораторы

Django предоставляет несколько декораторов, которые могут быть применены к функции представления, для поддержки различных функций HTTP. Декораторы представлений(функций-обработчиков) описаны [здесь](http://djbook.ru/rel1.8/topics/http/decorators.html).

Декораторы рассмотрим на примере решения одного из дополнительных домашних заданий, где нужно было ограничить доступ к админке, оставив доступ только администратору сайта.

Администратор отличается от обычного пользователя флагом .is\_superuser = True. Чтобы любого пользователя сделать администратором, нужно установить этот атрибут в True.

Декоратор user\_passes\_test() принимает обязательный аргумент: функцию, которая принимает объект User и возвращает True, если пользователю разрешён доступ к просмотру страницы. Следует отметить, что декоратор user\_passes\_test() не выполняет автоматическую проверку того, что User прошёл авторизацию.

Для нашей задачи декоратор примет вид: @user\_passes\_test(lambda u: u.is\_superuser)

Чтобы закрыть доступ к админке всем, кроме супер пользователя:

|  |
| --- |
| @user\_passes\_test(lambda u: u.is\_superuser)  def admin\_page(request):  # TODO: сделать доступ к админке только суперпользователю  users = User.objects.all()  user\_form = MyRegistrationForm()  return render(request, 'admin\_page.html', {'users': users, 'form': user\_form})  @user\_passes\_test(lambda u: u.is\_superuser)  def admin\_gems(request):  gems = User.objects.all()  return render(request, 'admin\_gems.html', {'gems': gems}) |

## Собственные шаблонные фильтры

Мы уже пользовались встроенными шаблонными фильтрами и тегами. Django позволяет очень просто создавать собственные. По сути, шаблонный фильтр это просто функция python, описанная по определённым правилам.

### Теория

Собственные теги и фильтры шаблонов должны определяться в приложении Django. Если они логически связаны с каким-то приложением, есть смысл добавить их в это приложение, иначе создайте новое приложение.

Приложение должно содержать каталог templatetags на том же уровне, что и models.py, views.py и др. Если он не существует, создайте его. Не забудьте создать файл \_\_init\_\_.py, чтобы каталог мог использоваться как пакет Python.

Ваши теги и фильтры будут находиться в модуле пакета templatetags. Название модуля будет использоваться при загрузке библиотеки в шаблоне, так что убедитесь, что оно не совпадает с названиями библиотек других приложений.

В шаблоне вы будете использовать:

|  |
| --- |
| {% load poll\_extras %} |

Приложение, содержащее собственные теги и фильтры, должно быть добавлено в INSTALLED\_APPS, чтобы тег {% load %} мог загрузить его. Это сделано в целях безопасности.

Не имеет значения сколько модулей добавлено в пакет templatetags. Помните. что тег {% load %} использует название модуля, а не название приложения.

Библиотека тегов должна содержать переменную register, равную экземпляру django.template.Library, в которой регистрируются все определенные теги и фильтры. Так что в начале вашего модуля укажите следующие строки:

### Практическая часть урока

Добавим библиотеку фильтров в приложение MainApp  
MainApp/  
 templatetags/  
 my\_tags.py

Напишем демонстрационный фильтр в my\_tags.py

|  |
| --- |
| from django import template  register = template.Library()  def through\_one(string):  """  Переводит каждый нечетный символ в верхний регистр, а четный в нижний  Mother --> mOtHeR  """  new\_string = ''  for i in range(len(string)):  new\_string += string[i].upper() if i % 2 else string[i].lower()  return new\_string  register.filter('through\_one', through\_one)  if \_\_name\_\_ == '\_\_main\_\_':  print(through\_one('mmmmmmmmm') |

Не забываете, что это обычный код python, и функции можно (даже нужно) тестировать отдельно.

Используем на выводе имен драгоценных камней   
gems.html

|  |
| --- |
| {% extends "base.html" %}  {% block title %}  Gems  {% endblock %}  {% block content %}  **{% load my\_tags %}**  <div class="row">  {% for gem in gems %}  <div class="gem\_container">  <div class="name">  {{ gem.name| **through\_one** }}  </div>  <div class="image">  <img src="/static/images/{{ gem.image | default:"dfav.png" }}">  </div>  {{ gem.category.name }}  </div>  {% endfor %}  </div>  {% endblock %} |

# Подведем итоги

В месячном курсе невозможно охватить даже половину возможностей фрэймворка Django. Это очень мощный и постоянно развивающийся инструмент. Целью текущего курса было именно дать основные понятия, показать принцип работы и философию django. Надеюсь, мне это удалось.

Если вы поняли, насколько django упрощает жизнь веб-разработчика, продолжайте изучать его. Как показывает практика, простое чтение документации - малоэффективно, гораздо полезнее параллельно с чтением делать небольшие собственные проекты.

# 

# 

# Домашнее задание

Несмотря на то, что это последнее занятие, я продолжу помогать с доработкой ваших проектов. Пишите в комментариях ваши вопросы, присылайте задания на проверку.

Последняя порция заданий:

1. Изменить структуру проекта в соответствии с рекомендациями, полученными на этом уроке. Разбить проект на логические части. Проверить, что ничего не сломалось.
2. Продолжать работать над визуальным оформлением. Лучшие работы (с разрешения их авторов) будут использоваться для демонстрации следующим поколениям слушателей вебинара.
3. \* Попробовать самостоятельно реализовать поиск по сайту.

Успехов!

# Дополнительные материалы

Все то, о чём сказано в методичке, но подробнее:

1. [Приложения (apps)](http://www.djbook.ru/rel1.7/ref/applications.html)
2. [Менеджер URL’ов](http://djbook.ru/rel1.7/topics/http/urls.html#naming-url-patterns)
3. [URLS include](http://djbook.ru/ch08s02.html)
4. [Декораторы представлений](http://djbook.ru/rel1.8/topics/http/decorators.html)
5. [Собственные шаблонные фильтры и теги](http://djbook.ru/rel1.4/howto/custom-template-tags.html#custom-template-tags-and-filters)

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. [Django Book(rus)](http://djbook.ru/)
2. [Собственные шаблонные фильтры и теги](http://djbook.ru/rel1.4/howto/custom-template-tags.html#custom-template-tags-and-filters)
3. [Менеджер URL’ов](http://djbook.ru/rel1.7/topics/http/urls.html#naming-url-patterns)