Знакомство с Django

Поскольку Django был разработан в бурлящей среде новостного агентства, он призван сделать веб-разработку простой и быстрой. Далее представлен вводный обзор того, как можно писать веб-приложения при помощи Django.

Цель данного документа: предоставить вам достаточное количество технической информации для понимания основ работы Django, но следует иметь ввиду, что этот обзор не является учебником или справочником – хотя у нас всё это есть! Когда вы будете готовы к созданию своего первого проекта, вы можете начать с учебного руководства*: «Создаем свое первое приложение с Django, части1-7».*

Проектирование модели

Хотя вы вполне можете обойтись без использования базы данных, Django предоставляет object-relational mapper для описания формата БД в виде кода на языке Python.

Синтаксис моделей данных предлагает множество различных способов представления ваших данных, что помогает решать стандартные задачи по использованию БД. Вот небольшой пример:

```
mysite/news/models.py
from django.db import models

class Reporter(models.Model):
    full_name = models.CharField(max_length=70)

    def __str__(self):              # __unicode__ on Python 2
        return self.full_name

class Article(models.Model):
    pub_date = models.DateField()
    headline = models.CharField(max_length=200)
    content = models.TextField()
    reporter = models.ForeignKey(Reporter, on_delete=models.CASCADE)

    def __str__(self):              # __unicode__ on Python 2
        return self.headline
```

Настройка базы данных

Теперь выполните команду для автоматического создания таблиц базы данных:

```
$ python manage.py migrate
```

Команда migrate ищет все имеющиеся у вас модели и создает на их основе недостающие таблицы в БД. Также при необходимости предоставляет более продвинутый контроль над изменениями в структуре данных.

Наслаждайтесь свободным API

Дело в том, что вам доступен развитый и бесплатный Python API для доступа к вашим данным. API создаётся на лету, без предварительной генерации кода:

```
# Import the models we created from our "news" app
>>> from news.models import Reporter, Article

# No reporters are in the system yet.
>>> Reporter.objects.all()
[]
# Create a new Reporter.
>>> r = Reporter(full_name='John Smith')

# Save the object into the database. You have to call save() explicitly.
>>> r.save()

# Now it has an ID.
>>> r.id
1
# Now the new reporter is in the database.
>>> Reporter.objects.all()
[<Reporter: John Smith>]

# Fields are represented as attributes on the Python object.
>>> r.full_name
'John Smith'
# Django provides a rich database lookup API.
>>> Reporter.objects.get(id=1)
<Reporter: John Smith>
>>> Reporter.objects.get(full_name__startswith='John')
<Reporter: John Smith>
>>> Reporter.objects.get(full_name__contains='mith')
<Reporter: John Smith>
>>> Reporter.objects.get(id=2)
Traceback (most recent call last):
    ...
DoesNotExist: Reporter matching query does not exist.

# Create an article.
>>> from datetime import date
>>> a = Article(pub_date=date.today(), headline='Django is cool',
...     content='Yeah.', reporter=r)
>>> a.save()

# Now the article is in the database.
>>> Article.objects.all()
[<Article: Django is cool>]

# Article objects get API access to related Reporter objects.
>>> r = a.reporter
>>> r.full_name
'John Smith'

# And vice versa: Reporter objects get API access to Article objects.
>>> r.article_set.all()
[<Article: Django is cool>]
# The API follows relationships as far as you need, performing efficient
# JOINs for you behind the scenes.
# This finds all articles by a reporter whose name starts with "John".
>>> Article.objects.filter(reporter__full_name__startswith='John')
[<Article: Django is cool>]
# Change an object by altering its attributes and calling save().
>>> r.full_name = 'Billy Goat'
>>> r.save()

# Delete an object with delete().
>>> r.delete()
```

Динамический административный интерфейс: это не просто строительный материал – это готовый дом

Как только вы определите свои модели, Django может автоматически создать удобный интерфейс администратора, позволяющий авторизованным пользователям добавлять, удалять и изменять объекты. Для этого следует зарегистрировать свою модель:

mysite/news/models.py

```
from django.db import models



class Article(models.Model):

    pub_date = models.DateField()

    headline = models.CharField(max_length=200)

    content = models.TextField()

    reporter = models.ForeignKey(Reporter, on_delete=models.CASCADE)
```

mysite/news/admin.py

```
from django.contrib import admin

from . import models

admin.site.register(models.Article)
```

Задумка заключается в том, что редактировать разделы могут служащие или клиенты, или, может быть, только вы – и при этом вам не хотелось бы иметь дело с созданием бэкенд интерфейсов (backend interfaces) только лишь для управления контентом.

Обычным рабочим моментом при создании сайта на Django, является написание моделей и получение прав администратора для того, чтобы как можно скорее приступить к работе. Так ваши сотрудники (или клиенты) могут сразу же начать наполнение сайта данными, а вы параллельно продолжите подготавливать сайт для публикации.

Проектирование схемы URL

Красивая, элегантная схема URL является важной составляющей высококачественного веб-приложения. Django поощряет создание красивых схем URL и не захламляет их мусором, подобным .php или же .asp.

Проектируя модель URL для своего приложения, вы создаёте модуль Python, называемый менеджером URL-ов. Оглавление вашего приложения представляет из себя простое соответствие между URL-адресом и вызываемой функцией на Python. URLconfs служит для выделения URL адресов из кода на Python.

Вот таким образом может выглядеть схема URL для классов Reporter/Article, представленных выше:

```
mysite/news/urls.py
from django.conf.urls import url

from . import views

urlpatterns = [
    url(r'^articles/([0-9]{4})/$', views.year_archive),
    url(r'^articles/([0-9]{4})/([0-9]{2})/$', views.month_archive),
    url(r'^articles/([0-9]{4})/([0-9]{2})/([0-9]+)/$', views.article_detail),
]
```

Указанная схема URL содержит простые регулярные выражения для вызова соответствующей функции Python (соответствующего представления, “views”). Регулярные выражения используют скобки для “захвата” значения из URL-адреса. Когда пользователь запрашивает страницу, Django проходит по всем шаблонам по порядку и останавливается на первом подходящем (если ни один из шаблонов не подошёл, Django вызовет исключение 404.) Это происходит мгновенно, поскольку регулярные выражения компилируются ещё во время загрузки.

После того, как соответствующий шаблон найден, Django импортирует и вызывает представление, которое является по сути простой функцией на Python. Каждое представление передаёт объект запроса – он содержит возвращаемые метаданные – и значения, захваченные регулярным выражением.

К примеру, если пользователь запросил URL “/articles/2005/05/39323/”, Django должен вернуть функцию news.views.article_detail(request, '2005', '05', '39323').

Написание представлений

Каждое представление отвечает за выполнение одной из двух вещей: возвращение объекта HttpResponse, представляющего содержимое запрашиваемой страницы, или вызов исключения такого как Http404. Остальное зависит от вас.

Как правило, представление извлекает данные в соответствии с заданными параметрами, загружает шаблон и отображает этот шаблон вместе с полученными данными. Это пример представления year_archive, представленного выше:

```
mysite/news/views.py
from django.shortcuts import render

from .models import Article

def year_archive(request, year):
    a_list = Article.objects.filter(pub_date__year=year)
    context = {'year': year, 'article_list': a_list}
    return render(request, 'news/year_archive.html', context)
```

В примере используется язык шаблонов Django, который имеет несколько сильных особенностей, но вместе с тем остаётся простым в использовании даже для не-программистов.

Код, представленный выше, возвращает шаблон news/year_archive.html.

Django включает пути поиска по шаблонам, что позволяет свести к минимуму избыточность html-кода. В настройках Django нужно указать список каталогов, откуда Django сможет загружать шаблоны. Это делается с помощью настройки DIRS. Если шаблон не будет найден в первом каталоге, Django проверит второй и все последующие.

Скажем, шаблон news/year_archive.html был найден. Вот как он может выглядеть:

```
mysite/news/templates/news/year_archive.html
{% extends "base.html" %}

{% block title %}Articles for {{ year }}{% endblock %}

{% block content %}
<h1>Articles for {{ year }}</h1>

{% for article in article_list %}
    <p>{{ article.headline }}</p>
    <p>By {{ article.reporter.full_name }}</p>
    <p>Published {{ article.pub_date|date:"F j, Y" }}</p>
{% endfor %}
{% endblock %}
```

Переменные всегда окружены двумя фигурными скобками. {{ article.headline }} означает “вывести переменную, хранящую заголовок статьи.” Хотя точка используется не только для доступа к атрибутам: с её помощью можно провести поиск в словаре, поиск индекса и вызов функции.

Примечательно, что {{ article.pub_date|date:"F j, Y" }} использует Unix-стиль “pipe” (символ вертикальной черты “|”). Это называется шаблонным фильтром и это хороший способ для фильтрации значений переменной. В примере фильтр даты приводит объект datetime к нужному виду (как нахождение функции даты в PHP). Вы можете последовательно применить столько фильтров, сколько вам будет нужно. Вы можете писать собственные шаблонные фильтры. Вы можете писать собственные шаблонные теги, которые будут неявно выполнять ваш код.

Наконец, Django использует концепцию “наследования шаблонов”: это достигается путём включения базового шаблона {% extends "base.html" %}. И это значит, что первым будет загружен шаблон ‘base’, в котором определено множество различных блоков, которые в свою очередь также могут содержать в себе какие-то блоки. Короче говоря, это позволяет намного сократить количество html-шаблонов страниц: каждый новый шаблон должен включать уникальные элементы, присущие только ему.

Базовый шаблон “base.html”, включающий использование статических файлов, может выглядеть следующим образом:

```
mysite/templates/base.html
{% load staticfiles %}
<html>
<head>
    <title>{% block title %}{% endblock %}</title>
</head>
<body>
    <img src="{% static "images/sitelogo.png" %}" alt="Logo" />
    {% block content %}{% endblock %}
</body>
</html>
```

Проще говоря, он определяет внешний вид сайта (подгружает логотип), а также оставляет место для заполнения блоков в дочерних шаблонах. Это позволяет свести редизайн всего сайта к изменению единственного файла – базового шаблона.

Также этот подход позволит вам создавать несколько версий сайта, использующих различные базовые шаблоны, в то время как дочерние шаблоны останутся неизменными. Создатели Django использовали этот подход для написания отдельного шаблона для мобильных устройств – просто создав новый базовый шаблон.

Обратите внимание на то, что вы можете не использовать язык шаблонов Django, если отдаёте предпочтение какому-то иному. Хотя язык шаблонов Django как нельзя лучше подходит при создании моделей Django, никто не заставляет вас использовать его. Если уж на то пошло, знайте, что вы даже вправе не пользоваться API баз данных Django. Вы можете использовать другой уровень абстракции базы данных, вы можете читать XML-файлы, вы можете читать файлы с диска, да и вообще делать всё, что угодно. Каждая часть Django – модели, представления, шаблоны – отделены друг от друга.

Следующими вашими шагами будут *установка Django* и *чтение глав: «Создаём своё первое приложение с Django, части 1-7»*.