ДЗ#3: Работа с моделями и объектами моделей

Основы веб-разработки (первый семестр)

Здесь - та инфа, которая вам будет нужна по моделям Django для подготовки третьего ДЗ

Правильный импорт settings

В ходе подготовки к ДЗ вам потребуется настройка AUTH_USER_MODEL, которую нужно будет брать из настроек проекта. Настройки проекта импортируются так:

```
from django.conf import settings
```

При этом импортируется не просто файлик settings.py в исходном нетронутом виде, django дополнительно дозаполняет его дефолтными настройками

Использование русских букв в скриптах на python

В случае использования русских букв в начало файла (первой строкой) добавляется:

```
# coding: utf-8
```

Консоль django

Чтобы всяко поиграться с выборками, проверить какие-то гипотезы, итп - можно использовать консоль django

manage.py shell

Данная команда открывает терминал python, и импортирует в нём полностью всё окружение, необходимое для работы проекта (то же самое, что делает django-сервер перед стартом).
Внутри можно импортировать уже нужные модели, контроллеры и всё прочее, ну

Прямой эфир

Марина Даньшина 17 минут назад
Расписание занятий → Расписание пересдач 0

Екатерина Рогушкова 8 часов назад
Получение значков достижений на портале (ачивки) 2

Ольга Августан Вчера в 18:41
Опросы → Получение значков достижений на портале (ачивки) 2

Ольга Августан Вчера в 13:19
Общие вопросы → Завершение семестра: последние итоговые встречи 0

Ольга Августан Вчера в 13:12
Завершение семестра: итоговые встречи по дисциплинам 10

Марина Даньшина 25 Мая 2016, 14:01
Разработка на Java (первый семестр) → Забыли зонтик 0

Ksenia Sternina 25 Мая 2016, 13:38
Домашнее задание - Юзабилити-тестирование 4

Летяго Владимир 25 Мая 2016, 01:40
Разработка приложений на Android - 1 (второй семестр) → Контрольный рубеж 0

Летяго Владимир 24 Мая 2016, 22:45
Домашнее задание №3 10

Вячеслав Ишутин 24 Мая 2016, 22:31
Разработка на C++ (открытый курс) → Итоговая встреча по дисциплине 0

Вячеслав Ишутин 24 Мая 2016, 22:22
Разработка на C++ (открытый курс) → Итоговое занятие. Презентация курсовых проектов. 0

Александр Агафонов 24 Мая 2016, 19:17
Мастер-класс Ильи Стыценко онлайн «Использование OAuth 2 в приложениях на Django» 1

Марина Даньшина 23 Мая 2016, 20:59
Мероприятия → Мастер-класс Ильи Стыценко онлайн «Использование OAuth 2 в приложениях на Django» 1

Ольга Августан 23 Мая 2016, 17:56
Разработка приложений на iOS - 1 (второй

и интерактивно всё это опробовать.

ipython

Это пакет на python, сам по себе являющийся более удобным python-терминалом (присутствует autocompletion по "TAB", история команд, итп).
Ставим его с помощью pip install ipython, после чего manage.py shell начнёт автоматом его подхватывать.

manage.py dbshell

Если кому-то понадобится - данная команда берёт все нужные настройки соединения с БД из settings.py и вываливает вас в терминальный клиент базы данных, где уже на языке SQL можно пообщаться с БД напрямую.

Аттач файлов и картинок к модели

Для этого есть models.FileField и models.ImageField

```
class MyModel(models.Model):
    image = models.ImageField()
    attached_file = models.FileField()
```

ImageField и Pillow

Для работы с картинками Django требуется пакет Pillow, а ему, в свою очередь, требуются некоторые системные пакеты.
Поэтому в целом на ubuntu это выглядит так:

```
sudo apt-get install python-dev libjpeg8-dev zlib1g-dev
pip install Pillow
```

Не забываем сделать pip freeze, взять оттуда установленную версию Pillow и добавить в requirements.txt
Тем, у кого mac - brew install libtiff libjpeg webp little-cms2
Остальным придется погуглить что-то типа "Pillow requirements on ..."

settings.py

Чтобы показать ссылки на файл или картинку в модели - используем .url:

```
<img src="{{ post.image.url }}">
```

семестр) → Завершение семестра 0

Ольга Августан 23 Мая 2016, 17:16
Итоговая встреча по дисциплине 2

Ольга Августан 23 Мая 2016, 17:07
Основы веб-разработки (первый семестр) → Итоговая встреча по дисциплине 0

Ольга Августан 23 Мая 2016, 17:06
Основы мобильной разработки (первый семестр) → Итоговая встреча по дисциплине 2

Ольга Августан 23 Мая 2016, 17:05
Проектирование интерфейсов мобильных приложений (второй семестр) → Итоговая встреча по дисциплине 0

Ольга Августан 23 Мая 2016, 17:02
Проектирование СУБД (второй семестр) → Итоговая встреча по дисциплине 0

Ольга Августан 23 Мая 2016, 16:59
Общие вопросы → Завершение семестра: итоговые встречи по дисциплинам 10

Весь эфир

Блоги

Основы веб-разработки (первый семестр) 26,04
Разработка на C++ (открытый курс) 16,97
Основы мобильной разработки (первый семестр) 14,70
Разработка на Java (первый семестр) 13,58
Общие вопросы 5,71
Разработка приложений на Android - 1 (второй семестр) 5,66
Стажировка 4,52
Проектирование интерфейсов мобильных приложений (второй семестр) 3,42
Разработка приложений на iOS - 1 (второй семестр) 2,29
Проектирование СУБД (второй семестр) 2,26

А чтобы это работало - нужно в settings.py прописать IMAGE_URL и IMAGE_ROOT. Первое - это ссылка на локейшн в nginx, который раздает загруженные пользователями файлы (у нас это кажется было /media/). Второе - это путь к папке, в которую django должна складывать загруженные файлы (по аналогии со STATIC_ROOT).
Например:

MEDIA_URL = '/media/'
MEDIA_ROOT = '/home/vagrant/vdata/stackoverflow/media/'

Полезные методы у объекта модели

__unicode__

Метод __unicode__ есть в python буквально для всего, и вызывается тогда, когда необходимо получить представление объекта в виде строки. То есть, если вы просто печатаете что-либо через print - вызывается именно метод __unicode__ у переменной, которую вы печатаете. Если админка django показывает список объектов - вызывается он же. В шаблоне, когда вы просто выводите переменную ({{ post }}) - тоже вызывается он.
Метод __unicode__ должен возвращать переменную в строковом представлении.
Ближайший пример:

```
class M(models.Model):
title = models.CharField()
def __unicode__(self):
return self.title
```

Если необходимо, можно отдавать и более комплексный результат в __unicode__.
Например:

```
class M(models.Model):
title = models.CharField()
created_at = models.DateTimeField()
def __unicode__(self):
return u'{} от {}'.format(self.title, self.created_at)
```

Meta

Класс Meta содержит некоторую необязательную информацию о модели в целом.

Например:

verbose_name, verbose_name_plural

Данные аттрибуты использует админка Django, чтобы выяснить, как показывать название модели в единственном и множественном числе. Содержат строки.

ordering

Способ сортировки списка моделей по умолчанию. Является список из названия полей.

Например:

```
class Article(models.Model):
title = models.CharField()
created_at = models.DateTimeField()
class Meta:
verbose_name = u'Новость'
verbose_name_plural = u'Новости'
ordering = ('-created_at', )
```

Теперь в админке данная модель будет называться "Новость", в списках - "Новости", по умолчанию мы всегда будем получать выборку, отсортированную по дате создания в обратном порядке (за это отвечает "-" перед названием поля

QuerySet, objects, выборка объектов из БД

QuerySet - это метод в Django, отвечающий за представление запроса к базе данных. Когда мы создаем объект в базе - используется QuerySet, когда выбираем из базы список объектов - используется он же. Всегда при работе с базой используется QuerySet.
Кроме того, у каждой модели есть т.н. Manager - это объект, который вызывается по умолчанию при необходимости выдрать объекты из базы. Он лежит в аттрибуте objects и возвращает QuerySet по умолчанию для данной конкретной модели.

Пример:

Article.objects.all() - формируем запрос на выборку всех объектов Article из БД

QuerySet - Lazy object. То есть, само по себе обращение к QuerySet объекты из базы не вытаскивает, а лишь формирует внутри себя необходимые структуры для

выполнения запроса (фильтры, сортировки, различные маппинги полей и тому подобное).
Непосредственно обращение к базе данных происходит при явном вызове. Например, при попытке напечатать QuerySet с помощью print - произойдет запрос к базе данных. При попытке итерироваться по объектам в QuerySet (например, в шаблоне через {% for %}) - произойдет запрос.

Пример:

q = Article.objects.all() - БД не трогаем, лишь сформировали запрос
q = q.filter(title="lalala") - БД всё еще не трогаем, добавили в запрос фильтр по полю title
q = q.order_by('created_at') - БД всё еще не трогаем, добавили в запрос сортировку по полю created_at

А вот сейчас уже запросы к БД
print q
l = list(q)
for obj in q: print obj

Как видно из предыдущего примера, некоторые методы QuerySet возвращают его же, и таким образом могут быть объединены в цепь. То есть, вполне валидно вот так

q = Article.objects.filter(title="lalala").order_by('created_at')[:10] - отсортировали по полю title, упорядочили по полю created_at и взяли десять первых объектов из запроса

filter

QuerySet можно фильтровать множеством различных способов, для чего используется метод filter. На вход в качестве имен параметров подаются имена полей модели, в качестве значений - то, с чем мы хотим сравнить эти поля в запросе.
Кроме того, существует принятая в Django нотация через "__" (два подчеркивания).
Применяется она для двух различных случаев:

Для использования одного из фильтров Django

Есть возможность не просто делать выборку по принципу "возьмем все новости, у которых title равен чему-либо", но и более сложные штуки вроде "возьмем все новости, у которых дата создания меньше данной", "все, у которых количество просмотров больше определенного", "все, созданные в ноябре этого года", "все, у

которых в тексте есть слово 'Путин'" и тому подобное. Для этого к названию поле через __ присоединяется один из встроенных фильтров Django. Он описывает, какое именно сравнение мы хотим произвести между данным полем и значением. Лучше всего можно понять это на примере

Article.objects.filter(title="lalala") - все, где title РАВЕН lalala
Article.objects.filter(title__contains="lalala") - все, где title СОДЕРЖИТ lalala
Article.objects.filter(title__icontains="lAlAlA" - все, где title СОДЕРЖИТ lalala, независимо от регистра
Article.objects.filter(created_at__lt=datetime.datetime(2016, 3, 10)) - все объекты, созданные до 10 марта (поле created_at МЕНЬШЕ datetime.datetime(2016, 3, 10)
Article.objects.filter(likes_count__gte=50) - все, где поле likes_count БОЛЬШЕ-ЛИБО-РАВНО 50

Думаю, суть понятна.

Для сравнения полей связанных моделей

Представим, что у нас есть модель Post, и на него через ForeignKey ссылается модель Comment.
Тогда через comment__имя-поля мы можем фильтровать комментарии по значениям из связанной модели. На пример:

Comment.objects.filter(post__title="lalala") - все комментарии, где у связанного поста title равен lalala

Комбинирование фильтров и полей связанных моделей

Можно делать и это. Например:

Comment.objects.filter(post__likes_count__gt=50) - все комментарии, где у связанного поста поле likes_count больше 50

Кроме того, можно следовать по цепочке таких связок на сколько угодно моделей вперед. К примеру, пусть модель Post ссылается на модель Blog. Тогда можно:

Comment.objects.filter(post__blog__name="Фоточки") - все комментарии, привязанные к постам, которые лежат в блогах с полем name равным "Фоточки"

Фильтр по связанной модели

Допустим, у нас у модели Post есть автор (модель пользователя) в поле Author (author=models.ForeignKey(settings.AUTH_USER_MODEL).
И допустим, что у нас есть переменная request.user (текущий пользователь).
Тогда мы можем сделать так:

Post.objects.filter(author=request.user)
Post.objects.filter(author=request.user.id)

То есть, для ForeignKey фильтр работает и с объектом модели, и с id этого объекта.

order_by

Кроме того, база данных предоставляет нам возможность сортировать выборку по нужному полю (или нескольким полям последовательно). Django же дает нам воспользоваться этим с помощью метода order_by. На вход этого метод принимает имена полей в качестве аргументов, например:

Post.objects.order_by('title') - сортируем посты по заголовку
Post.objects.order_by('-title') - сортируем посты по заголовку в обратном порядке
Post.objects.order_by('created_at', 'title) - сортируем сначала по дате создания, а внутри одинаковых дат - по заголовку

С order_by можно вертеть точно такие же фокусы через "__", как и для метода filter. То есть, мы можем вот такое:

Comment.objects.order_by('post__title') - сортируем каменты по названию связанного поста
Comment.objects.order_by('post__author__username') - сортируем каменты по логины пользователя, который создал пост, в котором опубликован камент! :)

Срезы

Список моделей можно выдрать из БД не полностью, а только часть. Для этого применяется та же нотация срезов, что и в целом в python. Покажу на примере:

Blog.objects.all()[:10] - десять первых блогов из выборки
Blog.objects.all()[25:30] - выборка блогов с 25го по 30й

Важно понимать, что вообще без сортировки мы берем десять первых блогов просто в порядке создания (или, если угодно, считайте, что по умолчанию всё сортируется по id).
Срезы обычно используются для вывода объектов постранично (с 1 по 10, с 10 по 20, ну и так далее)

Комбинируем всё вместе

Просто комплексный пример

Comment.objects.filter(post__author=request.user).order_by('blog__title', '-likes_count')[:10]
Все комментарии к постам текущего пользователя, упорядоченные сначала по названию поста, а внутри одинаковых названий постов - по количеству лайков в порядке убывания, первые десять комментариев

Забирание одного конкретного объекта из БД

get

Метод get возвращает нам один объект из базы данных по определенному фильтру (формат такой же, как и у filter). Использует чаще всего фильтрация по id:

Post.objects.get(id=10) - хотим получить пост с id=10

В случае, если такого объекта в базе нет, или если таких больше одного - произойдет ошибка (Post.DoesNotExist, и еще какая-то, не помню :)
А значит, если вы делаете это внутри view - получите ошибку сервера. Как с этим справиться - читаем дальше

get_object_or_404

Это не метод QuerySet-а, а вспомогательная функция, лежит в django.shortcuts (кстати, там собраны и другие весьма полезные штуки).
Эта функция получает на вход QuerySet, и список параметров, аналогичных методу filter). И либо возвращает найденный объект, либо провоцирует возникновение ошибки Http404. Этот класс ошибок Django ловит самостоятельно в процесе выполнения запроса, и возвращает страницу "Не найдено" (а это намного лучше, чем "ошибка сервера"). Таким образом, внутри любого из этапов обработки запроса (грубо говоря, внутри view), мы можем сделать так:

from django.shortcuts import get_object_or_404
импортируем get_object_or_404:

и где-нибудь в коде view:

post = get_object_or_404(Post.objects.all(), id=10)
либо получим пост с id=10 в переменную post, либо пользователь увидит страницу "Не найдено"

а можем и так, например:

post = get_object_or_404(Post.objects.filter(author=request.user), id=10)
берем все посты, где текущий юзер является автором, ищем среди них один с id=10
страницу "Не найдено" пользователь получит, если а) он не является автором поста с id=10 б) такого поста вообще нет в базе

first, last

Допустим, нам пофигу, встречается ли такой объект в выборке один или несколько раз, или его вообще там нет. И допустим, что мы в случае отсутствия объекта вообще не хотим никаких ошибок, а просто хотим None. Запросто, за это и отвечают методы first и last.
Проще всего на примере:

Post.objects.filter(title="lalala").first() - получаем первый пост в выборке, у которого title равен "lalala"
Post.objects.filter(title="lalala").first() - аналогично, последний пост с заданными условиями

Естественно, можно комбинировать first и last с сортировкой и фильтрами:

Post.objects.filter(author=request.user).order_by('-like_count').first() - получаем пост текущего юзера с максимальным количеством лайков, или None, у него вообще нет постов

Создание и редактирование объектов в БД

Изменение аттрибутов моделей

Происходит предельно просто - все поля модели есть у каждого объекта этой модели, и равны значениям из базы данных. Их можно переопределять. Важно понимать, что в базе автоматически ничего в этом случае не сохраняется. Сохранение отредактированной информации инициируется специальным методом save

post = Post.objects.first() - взяли первый попавшийся пост из базы
print post.title - напечатали его заголовок
post.title = u"Я взрослый мужик, я хочу пива, а не писать статьи" - отредактировали заголовок у поста
В БАЗЕ ДАННЫХ НИЧЕГО НЕ ПОМЕНЯЛОСЬ ПОКА ЧТО
post.save() - пост отредактировался в базе данных

post = Post() - а вот взяли и просто создали объект поста, его пока нигде нет, кроме как у нас в коде
print post.title - в заголовке ничего нет, вернее там лежит значение, указанное в качестве defaul для поля в models.py
post.title = u"А с другой стороны, я высадил уже четыре банки и больше не лезет. Старею." - установили заголовок
post.save() - новый объект создался в базе данных

post = Post(title=u"А и пофиг, выпью пятую!") - создали объект поста сразу с предустановленным заголовком
В БАЗЕ ПОКА НИЧЕГО НЕ ПОМЕНЯЛОСЬ
post.save() - а вот сейчас сохранили новый пост в базу

save

Данный метод используется для сохранения объектов в базу (раскрыто примером выше). У нас может быть только что созданный объект, либо уже существующий в базе. Определяется это наличием id у объекта (аттрибут id равен либо None, либо какому-то числу). Если id пуст - создаем новый объект, если нет - редактируем существующий (тот, который в базе лежит с этим id).
В случае создания объекта к связанной модели - просто подставляем объект в соответствующее поле. Например, создадим камент к посту с id=10:

post = Post.objects.get(id=10) - выдрали из базы пост
comment = Comment(post=post, title=u"Я убер-камент, вы облайкаетесь!") - создали

объект камента с предзаполненными полями, в том числе и с полем post, которое теперь указывает на наш пост
comment.save() - сохранили

Связанные модели

обратная связь, related_name

По умолчанию, когда мы привязываем одну модель к другой, создается в том числе "обратная выборка" у модели, к которой мы что-то привязываем. То есть, например:

```
class Post(models.Model):
...

class Comment(models.Model):
post=models.ForeignKey('blogs.Post')
```

У нас не только в каждом комментарии будет поле post, в котором валяется тот пост, к которому привязан камент. В объекте поста появляется аттрибут comment_set, который является QuerySet-ом, заранее отфильтрованным по посту. То есть, post.comment_set.all() равнозначно такому - Comment.objects.filter(post=post) (и кстати, с ним можно творить всё, описанное выше - это такой же QuerySet).

Как видите, по умолчанию Django берет, имя модели, приводит в нижний регистр и добавляет _set. Такое поведение можно изменить, добавив параметр related_name. Например, если мы опишем поле как post=models.ForeignKey('blogs.Post', related_name='comments'), то у постов появится аттрибут comments вместо comment_set. Это красивее.

Кроме того, эта возможность пригодится вам, когда вы будете создавать несколько связей с одной и той же моделью. Например:

```
class Post(models.Model):
author = models.ForeignKey(settings.AUTH_USER_MODEL, related_name='posts')

class Comment(models.Model):
author = models.ForeignKey(settings.AUTH_USER_MODEL,
related_name='comments')
```

Вот в этом случае вы можете запросто сделать request.user.posts, request.user.comments - все посты или комментарии текущего пользователя.

Представим такую ситуацию - у вас есть модерация, и соответственно у поста есть не только автор, но и модератор, который пост опубликовал. А значит, вы имеете две связи поста с пользователем. Тогда вам придется задать related_name, иначе Django откажется создавать такую связь:

```
class Post(models.Model):
author = models.ForeignKey(settings.AUTH_USER_MODEL,
related_name='created_posts')
moderator = models.ForeignKey(settings.AUTH_USER_MODEL,
related_name='published_posts')
```

А у текущего пользователя вы сможете заиметь две выборки постов - user.created_posts (посты, опубликованные юзером) и user.published_posts (посты, которые юзер разрешил к публикации в качестве модератора).

Фильтр по обратной связи

Есть возможность, которая не всегда очевидна - фильтр модели по обратной связи с другой моделью. На примере:

Post.objects.filter(comments__title__icontains=u"Путин") - все посты, в которых есть хотя бы один комментарий, содержащий в заголовке слово "Путин" (кстати, кто-нибудь в курсе, кто это? часто про этого мужика слышу).

Имейте ввиду, в этом случае, если в каком-либо посте есть два таких комментария, то и пост вы получите в выборке два раза (особенность работы СУБД, на втором курсе вам про это расскажут лучше меня).
Чтобы избежать такой ситуации - у QuerySet есть метод distinct (выборка только записей, уникальных по какому-либо полю, грубо говоря исключение дубликатов из выборки).
То есть, выборка будет такой:

Post.objects.filter(comments___title__icontains=u"Путин").distinct("id") - выбираем то же, что и до этого, но следим, чтобы в выборке не было постов с одинаковым id

Справочное чтиво

Здесь https://docs.djangoproject.com/en/1.9/ref/models/fields/ сначала есть описание всех возможных аргументов для полей в models.py, потом - все возможные типы полей.
Здесь https://docs.djangoproject.com/en/1.9/ref/models/querysets/#id4 - все возможные фильтры для полей (а в целом там описание всех возможных методов QuerySet-а)
И тот и другой источник есть в разделе документации на http://djbook.ru/ на русском языке. Либо гугл в помощь :)

Пожалуй, пока всё. В комментариях можете задавать вопросы. Если вы их не задаёте, то я считаю, что по этой теме всем всё понятно :)
Если задаёте - дополняю пост, не нарушая существующего формата.