ДЗ#4: generic relations

Основы веб-разработки (первый семестр)

Исходные данные - у нас есть проект, в нем есть разные типы объектов (у меня это Question и Answer как минимум).
Задача - прикрутить в проект лайки таким образом, чтобы можно было использовать одну модель лайка и для вопросов и для ответов (и вообще для чего угодно). Заодно это будет неплохой опыт написания приложения, работающего standalone - так, чтобы можно было взять его целиком и перенести в любой другой проект.
Под катом описание того, как я это вижу.

Пусть весь функционал, относящийся к лайкам, лежит в приложении likes, в нём и будем работать.

Модель лайка

Итак, наша задача - хранить в базе данные о том, что такой-то пользователь лайкнул такой-то объект. Соответственно, модель лайка должна реализовывать связи минимум с двумя объектами - это "лайкнувший" пользователь, с одной стороны, и "лайкнутый" объект с другой.
Добавим сюда еще дату лайка (мало ли, захотим какой-нибудь график потом построить...). Получаем вот такой код модели:

```
from django.contrib.contenttypes.models import ContentType
from django.contrib.contenttypes.fields import GenericForeignKey

class Like(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL)
    created_at = models.DateTimeField(auto_now_add=True)
    item_type = models.ForeignKey(ContentType)
    item_id = models.PositiveIntegerField()
    item = GenericForeignKey('item_type', 'item_id')
```

Типы первых двух полей (привязка к пользователю и дата создания) нам знакомы, на третьем же сейчас остановимся чуть подробнее.

django.contrib.contenttypes

Данное приложение является частью инфраструктуры django и служит для интроспекции создаваемых моделей.
Основной приложения служит модель ContentType, которая выглядит вот так:

Прямой эфир

Марина Даньшина 21 минуту назад
Расписание занятий → Расписание пересдач 0

Екатерина Рогушкова 9 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

Связь с разработчиками

```
class ContentType(models.Model):
    app_label = models.CharField(max_length=100)
    model = models.CharField(_('python model class name'), max_length=100)
```

То есть, ContentType хранит информацию о: имени какой-либо модели и приложении, в котором эта модель зарегистрирована. Когда в проекте описывается новая модель, в таблице django_content_types добавляется запись об этом (во время migrate). Таким образом, для каждой модели в проекте django существует объект ContentType, однозначно её (модель) идентифицирующий. А значит, мы можем использовать ForeignKey на ContentType, чтобы хранить в базе **тип** объекта, к которому привязана какая-либо модель. Тогда нам останется лишь добавить id этого объекта в качестве еще одного поля, и у нас будет достаточно информации в базе, чтобы однозначно идентифицировать объект **любого** типа. Это мы и делаем в данных двух строчках:

```
    item_type = models.ForeignKey(ContentType)
    item_id = models.PositiveIntegerField()
```

item_type - это у нас связка с ContentType (то есть, с моделью как с типом объектов), а item_id - это просто число, id объекта того типа, который указан в item_type.
Например, я хочу привязать лайк к ответу с id=53.
Судя вот по этому коду (из manage.py shell):

```
In [16]: from django.contrib.contenttypes.models import ContentType

In [17]: ContentType.objects.get(model='answer', app_label='answers')
Out[17]: <contenttype: Вопрос>

In [18]: ContentType.objects.get(model='answer', app_label='answers').id
Out[18]: 8
```

У меня модели ответа соответствует ContentType с id=8. А значит, в моем случае в базе у этого лайка item_type будет равен 8, а item_id - 53:

```
In [20]: from likes.models import Like
In [21]: l = Like()
In [22]: l.item_type = ContentType.objects.get(model='answer', app_label='answers')
In [23]: l.item_id = 53
```

GenericForeignKey

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

GenericForeignKey - это не настоящее поле, а просто средство автоматизации.
При использовании GenericForeignKey мы можем заменить код, показанный мной выше, на вот такой:

```
In [26]: answer = Answer.objects.all()[0]

In [27]: l = Like()

In [28]: l.item = answer

In [29]: l.item_type
Out[29]: <contenttype: Вопрос>

In [30]: l.item_id
Out[30]: 6
```

Здесь мы просто берём первый попавшийся ответ из базы, и присваиваем его атрибуту item (который описан в модели как GenericForeignKey).
А GenericForeignKey уже за нас смотрит, что же конкретно ему пытаются всучить, и заполняет поля item_type и item_id. При доставании лайка из базы атрибут item заполнится из полей item_type и item_id, и там будет лежать наш тот самый вопрос, который мы туда до этого сохранили.

GenericRelation

Окей, допустим, мы знаем, как сохранить лайк к любой модели. Мы понимаем, для чего нужен GenericForeignKey, и можем для каждого лайка достать "лайкнутый" объект из атрибута item. Но что, если нам нужно получить все лайки объекта (то есть реализовать обратную связь)?
В случае с ForeignKey для этого служит атрибут related_name - мы могли указать у ответа question = models.ForeignKey(Question, related_name='answers') и впоследствии получить все ответы через .answers.all()
В случае с GenericForeignKey это не сработает, нужно определять обратную связь самостоятельно, для этого служит класс GenericRelation. Пример того, как он работает:

```
from django.contrib.contenttypes.fields import GenericRelation

class Question(models.Model):
    ...
    likes = generic.GenericRelation(Like, content_type_field='item_type', object_id_field=
```

То есть, мы указываем модель для связи, и имена полей, в которых мы храним

ContentType и id объекта. Тогда django сможет построить для нас обратную выборку, и мы сможем в каждом объекте вопроса пользоваться атрибутом likes, чтобы как минимум посчитать количество лайков для конкретного объекта через question.likes.count()

LikeMixin

Как видно из предыдущего пункта, для нормального пользования лайками нам нужно определять дополнительно GenericRelation на той модели, которую мы хотим лайкать. Копипастить атрибут на каждую модель - совсем не вариант, мы можем воспользоваться наследованием. Пропишем дополнительно в приложении likes примерно вот такой класс:

```
from django.contrib.contenttypes.fields import GenericRelation

class LikeMixin(models.Model):

    likes = GenericRelation(Like, content_type_field='item_type', object_id_field ='item_id

    class Meta:
        abstract = True # означает, что для этой модели не нужно создавать таблицу в
```

Теперь мы можем наследовать Question не от models.Model, а от нашего LikeMixin

```
from likes.models import LikeMixin

class Question(LikeMixin):
    ...
```

То же самое можем проделать и с остальными моделями, к которым хотим прикрутить лайки.

Итак

У нас есть модель лайка, и есть Mixin, который позволяет добавить в любую другую модель атрибут likes для получения всех лайков объекта.
Осталось реализовать сам процесс лайка

Процесс лайка

Собственно, процесс лайка разбивается на две логичные части:
1) Проверяем, не лайкал ли данный пользователь уже данный объект

2) Если не лайкал, создаем объект Like с данным пользователем и данным объектом, сохраняем его в базу
3) Возвращает новое количество лайков (или например редиректим на ту страницу, с которой лайк был произведен)

Как однозначно определить url для объекта любого типа

Допустим, у нас есть какой-либо LikeView в likes/views.py
Как мы уже видели, для однозначного определения объекта нам нужно знать id объекта, и id его ContentType-а.
А значит, нам подходит примерно вот такой паттерн в likes/urls.py (не забываем, что нужно еще подключить likes.urls в application.urls):

(в application/urls.py)

```
    ...
    url(r'^likes/', include('likes.urls', namespace='likes')),
    ...
```

(в likes/urls.py)

```
    url(
        r'^(?P<content_type_id>\d+)/(?P<pk>\d+)/like/$',
        login_required(views.LikeView.as_view()),
        name='do_like'
    ),
```

Таким образом, в методе dispatch у LikeView мы можем сделать нечто подобное:

```
from django.views.generic import View
from django.contrib.contenttypes.models import ContentType
from django.shortcuts import get_object_or_404


class LikeView(View):

    def dispatch(self, request, content_type_id=None, pk=None, *args, **kwargs):
        content_type = get_object_or_404(ContentType, id=content_type_id)
        model_class = content_type.model_class()
        self.object = get_object_or_404(model_class, id=pk)
        return super(LikeView, self).dispatch(request, *args, **kwargs)
```

Здесь мы сначала получаем объект модели ContentType по переданному content_type_pk. После чего у content_type вытаскиваем соответствующую модель. А уже с помощью нее получаем тот объект, который пользователь захотел лайкнуть.
Если вспомнить пример выше про "лайк к ответу с id=53" - то у нас здесь content_type_pk будет равен 8, а pk будет равен 53. А в переменной model_class будет лежать класс Answer, от которого мы уже и получаем вопрос с id=53.

Генерация ссылки для лайка

Итак, у нас есть LikeView, который что-то делает, и ссылка вида /likes/(content_type_id)/(pk)/like/, которую этот LikeView обрабатывает.
Наша задача теперь получить такую ссылку для любого объекта. И эта задача сводится в принципе к тому, чтобы получить нужный ContentType (ведь id объекта мы и так всегда знаем).
Я бы рекомендовал добавить метод в LikeMixin:

```
class LikeMixin(models.Model):

    likes = GenericRelation(Like, content_type_field='item_type', object_id_field ='item_id

    @models.permalink
    def get_like_url(self):
        content_type = ContentType.objects.get_for_model(self.__class__)
        return 'likes:do_like', (), {'content_type_id': content_type.id, 'pk': self.id}

    class Meta:
        abstract = True
```

Обратите внимание, здесь на метод get_like_url навешен декоратор models.permalink, который интерпретирует результат работы метода как имя паттерна в urls.py, список args и массив kwargs. Внутри же самого метода мы сначала забираем из базы нужный ContentType (в self.__class__ лежит модель для объекта, а get_for_model - это специальный метод для получения ContentType-а по модели). Потом нам остается лишь предоставить в return имя патеррна, и нужные аргументы, models.permalink превратит это в ссылку в соответствии с urls.py.
А значит, в любом шаблоне мы можем использовать что-то типа {{ question.get_like_url }} или {{ answer.get_like_url }} и получить таким образом ссылку, при переходе на которую произойдет лайк объекта.

Кстати, сам лайк я оставляю на реализацию вам. Как в LikeView получить сам объект - написано выше. А дальше - переопределяем например метод get и делаем там всё, что нужно, после чего либо возвращаем редирект, либо новое

количество лайков - у кого какая прописана логика :)
Можно, кстати, наследовать LikeView не от View, а от TemplateView - тогда сможете определить template_name, и вернуть кусок html-а. Вобщем, кому как нравится.

В шаблонах

Тут всё очень просто - определяем какой-нибудь likes/templates/likes/like_button.html, рисуем в нем те кнопки лайка, которые вам нужны.
После чего можем включить этот шаблон рядом с любым нужным объектом с помощью тега {% include %}.
Например, пусть в like_button.html у нас используется ссылка {{ object.get_like_url }}
И пусть у нас в текущем контексте есть вопрос в переменной current_question.
Тогда мы можем отрисовать кнопку лайка вот так - {% include "likes/like_button.html" with object=current_question %}
Естественно, мы можем в шаблоне использовать что-то типа {{ object.likes.count }} для получения текущего количества лайков.

Собственно, отрисовку кнопок и количества лайков я тоже оставляю вам, тут всё очень индивидуально.

Итак

Таким образом, мы получаем более-менее standalone-приложение для лайков, которое можно вполне просто подключить к любому проекту и оно просто начнет работать.
Его можно сделать еще более отдельным, если избавиться от LikeMixin в моделях. Если кто-то придумает, как именно, и реализует - накину еще 2 балла :)

Как обычно - вопросы в личку либо каментами.