

Лекция №15

Django (продолжение)

- Валидаторы
- Язык шаблонов Django
- Работа с моделями из представлений
- Формы
- Обработка POST-запросов
- Расширение шаблонов
- Использование CSS
- Практика

Валидаторы

Прежде чем приступить к продолжению, обновим нашу модель. По аналогии с другими полями создаем поле поле rate (оценка), используя класс IntegerField. При этом мы хотим ограничить возможную оценку от 1 до 10. Для этого при создании rate в конструктор класса IntegerField передаются валидаторы:

```
# Файл coolapp/models.py
from django.db import models
from django.core.validators import MaxValueValidator, MinValueValidator

class Film(models.Model):
    name = models.CharField(max_length=200)
    desc = models.TextField()
    pub_date = models.DateTimeField('date published', auto_now_add=True)
    rate = models.IntegerField(validators=[MinValueValidator(1),
                                         MaxValueValidator(10)], default=1)
```

Валидаторы передаются списком в аргумент `validators`, при этом в их конструкторы передаются сами граничные числа. Также при создании поля rate указываем аргумент `default` - это будет значение по умолчанию.

Миграция обновлений

После изменений в модели создаем миграцию и применяем ее:

```
$ python3 manage.py makemigrations coolapp  
$ python3 manage.py migrate
```

Язык шаблонов Django

В базе уже имеется несколько фильмов, но вновь созданное поле у них отсутствует. Это необходимо предусмотреть в html шаблоне (films.html), используя язык шаблонов (template language):

```
{% for film in films %}
<h2> {{ film.name }} </h2>
<p> {{ film.desc }} </p>
{% if film.pub_date %}
<p> Film date - {{ film.pub_date }}! </p>
{% else %}
<p> Film date - Unknown! </p>
{% endif %}
<p> {{ film.rate }} </p>
{% endfor %}
```

Работа с моделями из представлений

Для подстановки в шаблон объектов класса Film, на которые отображаются записи соответствующей таблицы базы данных надо дописать представление films в coolapp/views.py:

```
from django.shortcuts import render
from .models import Film

def index(request):
    return render(request, 'coolapp/index.html')

def films(request):
    return render(request, 'coolapp/films.html',
                  {'films': Film.objects.all()})
```

Формы (forms)

Половина функциональности сайта уже реализована. Еще один важный аспект, который стоит предусмотреть, - удобный способ добавления и редактирования записей. Панель admin Django удобна, но её дизайн сложно изменять. Формы (forms) позволяют создавать всевозможные компоненты интерфейса сайта и организовывать сам интерфейс.

В формах Django удобно то, что можно создать новую форму с нуля или воспользоваться `ModelForm` для сохранения содержимого форм в модель. Это как раз то, что нам нужно сделать - мы создадим форму для модели `Film`. Как и любая важная часть Django, формы имеют свой собственный `forms.py`. Нам нужно создать файл с таким именем в директории `coolapp`.

```
# Файл coolapp/forms.py
from django import forms
from .models import Film

class FilmForm(forms.ModelForm):
    class Meta:
        model = Film
        fields = ('name', 'desc', 'rate')
# поля pub_date и id заполняются сами
```

Формы (forms)

Для странички добавления фильма нужно определить свой url и добавить его в `coolapp/urls.py`.

```
# Файл coolapp/urls.py
from django.urls import path
from . import views
```

```
urlpatterns = [
    path('', views.index, name='index'),
    path('new/', views.new, name='new'),
    path('films/', views.films, name='films'),
]
```

Формы (forms)

Также потребуется отдельный view - new. Для этого нужно создать экземпляр объекта FilmForm и передать его в render под любым названием.

```
# Файл coolapp/views.py
from django.shortcuts import render
from .models import Film
from .forms import FilmForm

def index(request):
    return render(request, 'coolapp/index.html')

def films(request):
    return render(request, 'coolapp/films.html',
                  {'films': Film.objects.all()})

def new(request):
    return render(request, 'coolapp/new.html', {'form': FilmForm()})
```


Формы (forms)

Но у нас нет html шаблона под FilmForm. Разумеется, его тоже нужно создать. Чтобы заставить все это работать, нам потребуется сделать следующее (в файле new.html):

- отобразить форму, это можно сделать при помощи фигурных скобок `{{ form.as_p }}`
- строка выше должна быть обернута в HTML-теги `<form method="POST">...</form>`
- потребуется кнопка Save, которую мы реализуем при помощи HTML-кнопки: `<button type="submit">Save</button>`
- и наконец сразу после открытия тега `< form... >` мы должны добавить `{% csrf_token %}`, это очень важно, поскольку так мы делаем форму защищенной! Django выдаст предупреждение, если не указать токен.

```
<h1>New film</h1>
<form method="POST">{% csrf_token %}
{{ form.as_p }}
<button type="submit">Save</button> </form>
```

Формы (forms)

Теперь заходим на <http://127.0.0.1:8000/new/> и смотрим что получилось. Заодно пробуем создать какой-нибудь фильм и сохранить его. И посмотреть, как он отобразится на страничке списка фильмов. Как видите, ничего не произошло. Нужно сделать кое-что еще, чтобы новое представление заработало.

После отправки формы по нажатию кнопки мы возвращаемся к тому же представлению, но в этот раз с новыми данными в `request`, а точнее в `request.POST` (т.к. в HTML-файле, определение `<form>` имеет параметр `method="POST"`). Все поля формы теперь находятся в `request.POST`.

Обработка POST-запросов

Получается, что в представлении view нам нужно обработать две разные ситуации. Первая: когда мы только зашли на страницу и хотим получить пустую форму. Вторая: когда мы возвращаемся к представлению со всей информацией, которую мы ввели в форму. Таким образом, нам потребуется ввести условие (мы будем использовать условный оператор `if` для этой цели).

```
# файл coolapp/views.py
```

```
def new(request):  
    if request.method == "POST":  
        [...]  
    else:  
        return render(request, 'coolapp/new.html', {'form': FilmForm()})
```

Обработка POST-запросов

В `request.POST` будут лежать все данные, которые мы заполнили. Соответственно, вместо [...] с прошлого слайда можно использовать такой код:

```
form = FilmForm(request.POST)
```

Дальше мы проверим, корректна ли форма (все необходимые поля заполнены, сохранены только верные значения). Мы сделаем это при помощи `form.is_valid()` в условии `if`. Если форма правильная, то мы должны сохранить наш фильм с помощью `form.save()`

Обработка POST-запросов

Чтобы посмотреть, как выглядит фильм сразу после создания, необходимо добавить еще один импорт во views:

```
from django.shortcuts import redirect
```

Теперь мы можем сделать переадресацию на страницу фильма для созданной записи:

```
def new(request, film_id=None):  
    if request.method == "POST":  
        form = FilmForm(request.POST)  
        if form.is_valid():  
            film = form.save()  
            return redirect ('/{}'.format(film.id), film=film)  
    if film_id:  
        film = Film.objects.get(id=film_id)  
    else:  
        film = Film()  
    return render(request, 'coolapp/new.html',  
                  {'form': FilmForm(instance=film)})
```

Теперь можно посмотреть результат в браузере.

Обработка POST-запросов

Также надо добавить url для передачи в форму new фильмов с известным id (полученными при сохранении формы):

```
from django.urls import path
from . import views

urlpatterns = [
    path('new/', views.new, name='new'),
    path('films/', views.films, name='films'),
    path('', views.index, name='index'),
    path('<int:film_id>/', views.new, name='new'),
]
```

Расширение шаблонов

Допустим, требуется на КАЖДОЙ страничке этого сайта писать имя разработчика. Дублировать код каждый раз - плохо, любой программист должен избегать дублирования. Но тут нам на помощь приходит удобный инструмент, который в Django называется расширение шаблонов. Он позволяет переиспользовать куски кода по месту назначения, не копируя код повторно. Сначала нужно создать базовый шаблон base.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>About films</title>
</head>
<body>
  <h1><a href="/">Site about films</a></h1>
  {% block content %}
  {% endblock %}
</body>
</html>
```

Расширение шаблонов

Мы создали `block` - тег шаблона, позволяющий вставлять HTML-код этого блока в другие шаблоны, расширяющие `base.html`. Для этого в html-код других шаблонов, например, шаблона со списком фильмов, в начало файла надо дописать:

```
{% extends 'coolapp/base.html' %}  
{% block content %}
```

а в конец:

```
{% endblock content %}
```

Таким образом мы расширили шаблон `base.html` шаблоном со списком фильмов. Теперь заходим на страничку со списком фильмов. Если все сделано правильно, то мы должны увидеть, что написанное нами в `base.html` как бы "окружило" нашу страничку со списком фильмов. Также можно "окружить" любую другую нашу страничку, к примеру, страничку добавления фильмов.

Использование CSS

Пока что наш блог имеет весьма примитивное оформление, что необходимо исправить. Для этого будем использовать CSS - каскадные таблицы стилей (Cascading Style Sheets) – специальный язык, используемый для описания внешнего вида и форматирования сайта, написанного на языке разметки (таком, как HTML). Если HTML отвечает за содержание, то CSS за оформление. Очевидно, в этом случае так же существует много шаблонов оформления, которые остается только выбрать и правильно подключить

Для начала зададим цвет фона нашего сайта. Нам придется иметь дело со статическими файлами. Статическими файлами называются все файлы CSS и изображений, т.е. файлы, которые не изменяются динамически, их содержание не зависит от контекста запроса и будет одинаково для всех пользователей

Использование CSS

Создаем папку `static` в `coolapp`, в ней папку `css`, в ней файл `main.css` со следующим содержанием:

```
html {  
    background-color: #faebd7; /* Цвет фона */  
}
```

В `base.html` допишем первой строкой:

```
{% load staticfiles %}
```

Внутри тега `head` добавим:

```
<link rel="stylesheet" href="{% static 'css/main.css' %}">
```

И снова запускаем сайт, открыв страничку, которую мы ранее расширили с помощью `base.html`.

Практика

1. Таблицы реляционной базы данных могут быть связаны между собой отношениями. Необходимо создать еще одну таблицу (например, комментарии) и связать ее с таблицей фильмов, отобразить данные этой таблицы на шаблонах (например, на страничке конкретного фильма должны отображаться комментарии, привязанные к этому фильму), добавить форму для добавления записей в эту таблицу (например, на страничке конкретного фильма должна быть форма добавления комментариев к этому фильму).
2. * Сейчас наша страничка для создания фильмов доступна любому пользователю, что неправильно. Необходимо сделать так, чтобы ссылка на страничку добавления отображалась только для пользователей, которые вошли с учетной записью администратора (необходимо использовать объект `request` из `views`).