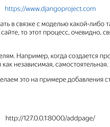
Использование форм, не связанных с моделями

Смотреть материал на видео

Архив проекта: lesson-13-coolsite.zip

На данный момент мы с вами, в целом, познакомились с механизмом маршрутизации, построения моделей, работы с шаблонами и использованием админ-панели. Конечно, это далеко не полный функционал фреймворка Django и мы еще будем возвращаться к этим темам, но на этом занятии мы коснемся новой темы – работы с формами.

Те из вас, кто уже создавал свои сайты, знают, что формы – это один из важнейших элементов большинства сайтов. Например, когда мы выполняем авторизацию или регистрацию, то появляется страница с полями ввода, чекбоксами, кнопками, списками и другими элементами интерфейса:



Это и есть формы. В HTML они задаются тегом <form> и служат для передачи на сервер пользовательской информации, например, логина и пароля для входа на сайт. На этом занятии мы увидим, как реализуются формы во фреймворке Django. Подробную документацию об этом вы можете почитать на русскоязычном сайте:

https://djbook.ru/rel3.0/index.html

в разделе «Формы», или на официальном англоязычном сайте:

https://www.djangoproject.com

Первое, что нужно знать, это то, что формы в Django можно создавать в связке с моделью какой-либо таблицы БД. Тогда получаем формы, связанные с моделью. Например, когда мы выполняем авторизацию или регистрацию на сайте, то этот процесс, очевидно, связан с данными таблиц, куда используются формы, связанные с моделями.

Но можно создавать и независимые формы, не привязанные к моделям. Например, когда создается простой поиск или идет отправка письма на электронную почту. Если при этом обращение к БД не требуется, то и форма создается как независимая, самостоятельная.

Сначала мы рассмотрим форму, не связанную с моделью, хотя и сделаем это на примере добавления статей в БД. На следующем занятии модифицируем ее и превратим в форму, связанную с моделью.

В главном меню у нас уже есть пункт для добавления статей:

http://127.0.0.1:8000/addpage/

и функция представления addpage (в файле women/views.py). Немного изменим эту функцию так, чтобы она отображала шаблон addpage.html:

```
def addpage(request):
    return render(request, 'women/addpage.html', {'menu': menu, 'title': 'Добавление статьи'})
```

А сам шаблон addpage.html определим так:

```
{% extends 'women/base.html' %}

{% block content %}
<h1>{{title}}</h1>
<p>Содержимое страницы
{% endblock %}
```

Теперь, при обновлении увидим полноценную страницу для добавления нового поста.

Создание класса формы

Все готово для создания формы. В Django существует специальный класс Form, на базе которого удобно создавать формы, не связанные с моделями. Где следует объявлять формы? Обычно, для этого создают в приложении отдельный файл forms.py. Мы так и сделаем (создаем файл women/forms.py). И в этом файле импортируем пакет forms и наши модели (модель Category нам здесь понадобится для формирования списка категорий):

```
from django import forms
from .models import *
```

Следующий шаг – объявить класс AddPostForm, описывающий форму добавления статьи. Он будет унаследован от базового класса Form и иметь следующий вид:

```
class AddPostForm(forms.Form):
    title = forms.CharField(max_length=255)
    slug = forms.SlugField(max_length=255)
    content = forms.CharField(widget=forms.Textarea(attrs={'cols': 60, 'rows': 10}))
    is_published = forms.BooleanField()
    cat = forms.ModelChoiceField(queryset=Category.objects.all())
```

Смотрите, мы здесь определяем только те поля, с которыми будет взаимодействовать пользователь. Например, поля модели time_create или time_update нигде не фигурируют, так как заполняются автоматически. Далее, каждый атрибут формы лучше назвать также, как называются поля в таблице women. Впоследствии нам это облегчит написание кода.

В классе формы каждый атрибут – это ссылка на тот или иной экземпляр класса из пакета forms. Например, title определяет через класс CharField, поле is_published – через BooleanField, а список категорий cat – через ModelChoiceField, который генерирует список из записей таблицы Category.

Почему указаны именно такие классы? И какие классы вообще существуют для формирования полей формы? Полный их список и назначение можно посмотреть на странице русскоязычной документации:

https://djbook.ru/rel3.0/ref/forms/fields.html

Я советую вам в целом изучить его и знать, как создавать различные типы полей. В частности, класс CharField служит для создания обычного поля ввода, класс BooleanField – для checkbox'а, класс ModelChoiceField – списка с данными из указанной модели.

Использование формы в функции представления

После того, как форма определена, ее можно использовать в функции представления addpage. В самом простом варианте можно записать так:

```
def addpage(request):
    form = AddPostForm()
    return render(request, 'women/addpage.html', {'menu': menu, 'title': 'Добавление статьи', 'form': form})
```

Здесь создается экземпляр формы и через переменную form передается шаблону addpage.html. Но это будет работать только при первом отображении формы, когда пользователь еще не заполнил ее поля. Если это есть, когда форма ассоциирована с данными. При повторном ее отображении, например, если данные были введены некорректно и нужно показать ошибки ввода, то форма должна сохранять ранее введенную пользователем информацию. Чтобы продолжать такой трюк, в функции представления пропишем следующее условие:

```
def addpage(request):
    if request.method == 'POST':
        form = AddPostForm(request.POST)
        if form.is_valid():
            print(form.cleaned_data)
    else:
        form = AddPostForm()

    return render(request, 'women/addpage.html', {'menu': menu, 'title': 'Добавление статьи', 'form': form})
```

Мы здесь вначале проверяем, если пришел POST-запрос, значит, пользователем были отправлены данные (мы будем передавать их именно POST-запросом). В этом случае напоминаем форму принятыми значениями из объекта request.POST и, затем, делаем проверку на корректность заполнения полей (метод is_valid). Если проверка прошла, то в консоли отобразим в словарь form.cleaned_data полученных данных от пользователя. Если же проверка не пройдет, то пользователь увидит сообщения об ошибках. Ну, а если форма показывается первый раз (идем по else), то она формируется без параметров и отображается с пустыми полями.

Отображение формы в шаблоне

Осталось отобразить форму в нашем шаблоне. Перейдем в файл addpage.html и пропишем там следующие строчки:

```
<form action="{% url 'add_page' %}" method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Добавить</button>
</form>
```

Смотрите, мы самостоятельно прописываем тег <form> для создания формы в HTML документе, указываем через атрибут action ее обработчик (в данном случае – это тот же адрес страницы, и связанная с ним функция представления addpage). Атрибут method указывает способ передачи информации на сервер (используется POST-запрос). Внутри формы добавляем специальный тег csrf_token, который генерирует скрытое поле с уникальным токеном. Это необходимо для защиты от CSRF-атак, когда на какой-либо другой сайте злоумышленник создает по виду неотличимую форму от вашего сайта и пытается заставить пользователя отправить актуальные данные на сервер через посредника <ul> или в виде таблицы. Последний вариант, хоть и возможен, но считается устаревшей практикой. Здесь также стоит иметь в виду, что по умолчанию все поля в словарь form.cleaned_data попадают только при указании обратное через параметр required=False.

Следующая строчка {{ form.as_p }} вызывает метод as_p нашей формы для отображения ее полей, используя теги абзацев <p>. Существуют и другие методы, которые формируют поля в виде элементов списка <ul> или в виде таблицы. Последний вариант, хоть и возможен, но считается устаревшей практикой. Здесь также стоит иметь в виду, что по умолчанию все поля в словарь form.cleaned_data попадают только при указании обратное через параметр required=False.

Наконец, последняя строчка – тег <button> создает кнопку типа submit для запуска процесса отправки данных формы на сервер и, в конечном итоге, нашей функции представления addpage.

Если теперь обновить страницу, то увидим все указанные поля формы со списком и кнопкой.

Улучшение внешнего вида формы

Но у нас название полей отображается по-английски и нам бы хотелось их изменить. Для этого у каждого класса полей формы есть специальный атрибут label, который и позволяет задавать свои имена, например, так:

```
class AddPostForm(forms.Form):
    title = forms.CharField(max_length=255, label="Заголовок")
    slug = forms.SlugField(max_length=255, label="URL")
    content = forms.CharField(widget=forms.Textarea(attrs={'cols': 60, 'rows': 10}), label="Контент")
    is_published = forms.BooleanField(label="Публикация")
    cat = forms.ModelChoiceField(queryset=Category.objects.all(), label="Категории")
```

Теперь, все выглядит гораздо приятнее. Давайте для примера сделаем поле content необязательным, а поле is_published с установленной галочкой. Соответственно, в классе CharField пропишем параметр required=False, а в классе BooleanField – параметр initial=True. Еще в классе ModelChoiceField добавим параметр empty_label="Категория не выбрана". Чтобы вместо черточек отображалась по умолчанию в списке эта фраза.

Со всеми возможными параметрами можно ознакомиться в документации, все по той же ссылке:

https://djbook.ru/rel3.0/ref/forms/fields.html

Способы отображения формы в шаблонах

Давайте теперь посмотрим, что же в действительности представляет собой объект form на примере ручного перебора и отображения всех наших полей. Я сейчас уберу строчку {{ form.as_p }} и вместо нее запишу все поля формы по порядку, друг за другом.

Первое поле title мы сформируем так:

```
<p><label class="form-label" for="{{ form.title.id_for_label }}">{{form.title.label}}: </label>{{ form.title }}
<div class="form-error">{{ form.title.errors }}</div>
```

Смотрите, мы здесь самостоятельно прописали HTML-тег внутри формы. Сначала идет тег абзаца <p>, внутри него тег <label> для оформления надписи. У нее указан класс оформления form-label и идентификатор через свойство form.title.id_for_label. Далее, идет само название form.title.label и после тега <label> отображается поле для ввода заголовка html. Это мы сформировали расписать атрибуты объекта form внутри шаблона. Ну а следующая строчка определяет тег <p> с классом оформления form-error для отображения возможных ошибок при вводе неверных данных. Список ошибок доступен через переменную form.title.errors.

Все, если теперь обновить страницу сайта, то увидим это одно поле в форме. По аналогии можно прописать и все остальные поля:

```
<p><label class="form-label" for="{{ form.slug.id_for_label }}">{{form.slug.label}}: </label>{{ form.slug }}</
<div class="form-error">{{ form.slug.errors }}</div>
<p><label class="form-label" for="{{ form.content.id_for_label }}">{{form.content.label}}: </label>{{ form.con
<div class="form-error">{{ form.content.errors }}</div>
<p><label class="form-label" for="{{ form.is_published.id_for_label }}">{{form.is_published.label}}: </label>{
<div class="form-error">{{ form.is_published.errors }}</div>
<p><label class="form-label" for="{{ form.cat.id_for_label }}">{{form.cat.label}}: </label>{{ form.cat }}</p>
<div class="form-error">{{ form.cat.errors }}</div>
```

А в самом верху добавим строчку:

```
<div class="form-error">{{ form.non_field_errors }}</div>
```

для вывода ошибок валидации, не связанных с заполнением того или иного поля.

Это довольно гибкий вариант представления формы и здесь можно очень тонко настроить отображение каждого поля. Однако, объем кода при этом резко возрастает. В большинстве случаев все это можно существенно сократить. Вы, наверное, уже заметили, что все эти строчки, в общем-то, повторяются, а значит, их можно сформировать через цикл, следующим образом:

```
<div class="form-error">{{ form.non_field_errors }}</div>

{% for f in form %}
<p><label class="form-label" for="{{ f.id_for_label }}">{{f.label}}: </label>{{ f }}</p>
<div class="form-error">{{ f.errors }}</div>
{% endfor %}
```

Я, думаю, этот код вполне понятен: мы здесь на каждой итерации имеем объект поля формы и также обращаемся к нужным его атрибутам. Такая запись значительно короче и гибче в плане изменения формы (достаточно поменять класс формы и это автоматом изменит ее вид в шаблоне). Правда, все поля теперь будут иметь один и те же стили оформления.

Однако, для виджетов стили оформлений можно прописать непосредственно в классе формы. Например, у класса поля ввода title добавить именованный параметр widget:

```
title = forms.CharField(max_length=255, label="Заголовок", widget=forms.TextInput(attrs={'class': 'form-input'
```

Мы здесь формируем виджет через класс TextInput и указываем у него стиль оформления form-input. При обновлении страницы, видим, что первое поле изменило свой вид. И так можно делать со всеми полями.

Тестирование формы

Если мы попробуем отправить пустую форму на сервер, то браузер укажет, что поле title обязательное. То же самое и для поля URL, напишем что-нибудь латинскими буквами. Выберем категорию и нажмем «Добавить». В результате, в консоли у нас отобразился словарь с принятыми данными:

{'title': 'Архана Гранде', 'slug': 'gfhjghjghjg', 'content': '', 'is_published': True, 'cat': <Category: Певицы>}

Все это мы можем сохранить в БД и сформировать новый пост. Если же данные в форме окажутся некорректными, например, в поле URL напишем что-то русскими буквами и сделаем отправку. Видим сообщение:

«Значение должно состоять только из латинских букв...»

и в консоли нет отображения данных, то есть, проверка не прошла. Вот так автоматически Django позволяет выполнять проверки на валидность заполненных данных.

Добавление новой записи

Теперь, когда наша форма в целом готова, выполним добавление записи в БД. Для этого после проверки валидности данных, запишем конструкцию:

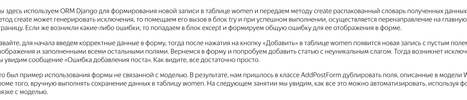
```
        if form.is_valid():
            #print(form.cleaned_data)
            try:
                Women.objects.create(**form.cleaned_data)
                return redirect('home')
            except:
                form.add_error(None, 'Ошибка добавления поста')
```

Мы здесь используем ORM Django для формирования новой записи в таблице women и передаем методу create распакованный словарь полученных данных. Так как метод create может генерировать исключения, то помещаем его вызов в блок try и при успешном выполнении, осуществляется перенаправление на главную страницу. Если же возникли какие-либо ошибки, то попадаем в блок except и формируем общую ошибку для ее отображения в форме.

Давайте, для начала введем корректные данные в форму, после нажатия на кнопку «Добавить» в таблице women появится новая запись с пустым полем для изображения и заполненными всеми остальными полями. Вернемся в форму и попробуем добавить статью с неучтенными слагом. Тогда возникнет исключение и мы увидим сообщение «Ошибка добавления поста». Как видите, все достаточно просто.

Это был пример использования формы не связанной с моделью. В результате, нам пришлось в классе AddPostForm дублировать поля, описанные в модели Women и, кроме того, вручную выполнять сохранение данных в таблицу women. На следующем занятии мы увидим, как все это можно автоматизировать, используя формы в связке с моделью.

Видео по теме



#1. Django - что это такое, порядок установки

#2. Модель MTV. Маршрутизация. Функции представления

#3. Маршрутизация, обработка исключений запросов.

#4. Определение моделей. Миграции: создание и выполнение

#5. CRUD - основы ORM по работе с моделями

#6. Шаблоны (templates). Начало

#7. Подкл статических Фильтры ш

← Предыдущая Следующая →