Формы в Django

Формы в Django — это мощный инструмент для обработки пользовательского ввода, валидации данных и генерации HTML-разметки

В Django выделяют два главных класса для работы с формами:

- ✓ forms.Form: Базовый класс для создания форм «с нуля». Используется, когда данные не нужно напрямую сохранять в базу данных (например, форма обратной связи или поиска).
- ✓ forms.ModelForm: Автоматически создает поля формы на основе существующей модели Django. Это значительно экономит время, так как Django сам подберет нужные типы полей и обеспечит сохранение данных в БД.

Ключевые возможности

- ✓ Автоматическая генерация HTML. Вы можете вывести всю форму в шаблоне одной командой, например **{{ form.as_p }} (как параграфы)** или **{{ form.as_table }} (как таблицу)**.
- ✓ Валидация (проверка) данных. Формы проверяют, корректно ли заполнены поля (например, является ли email валидным). Для этого используется метод is_valid().
- ✓ Очистка данных. После успешной проверки данные доступны в словаре **form.cleaned_data,** где они уже преобразованы в соответствующие типы Python (например, строка даты станет объектом datetime).

Формы в Django

По правилам архитектуры Django формы создаются в отдельном файле forms.py внутри конкретного приложения.

Для создания обратной связи не используется какая-либо модель, поэтому в этом случае нужно использовать forms.Form

myapp/forms.py:

```
from django import forms


class ContactForm(forms.Form):
    name = forms.CharField(label="Ваше имя", max_length=100)
    email = forms.EmailField(label="Email")
    message = forms.CharField(label="Сообщение", widget=forms.Textarea)
```

Свяжитесь с нами (forms.Form)

Экземпляр нужной формы нужно создать в представлении и передать её в контекст шаблона
myapp/views.py:

```python
from django.shortcuts import render
from django.core.mail import send_mail
from .forms import ContactForm

def contact_view(request):
    if request.method == "POST":
        form = ContactForm(request.POST) # Наполняем форму данными из запроса
        if form.is_valid():
            # Извлекаем очищенные данные
            subject = f"Сообщение от {form.cleaned_data['name']}"
            message = form.cleaned_data['message']
            sender = form.cleaned_data['email']
            # Отправка email (требует настройки в settings.py)
            send_mail(subject, message, sender, ['admin@example.com'])
            return render(request, 'success.html')
    else:
        form = ContactForm()
    return render(request, 'contact.html', {'form': form})
```

Django не создает тег <form> сам, его нужно прописать вручную. Форма внутри рендерится одной переменной form
myapp/templates/contacts.html

```
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Отправить</button>
</form>
```

Поля формы

1. Текстовые поля
   - CharField. Для любых коротких строк. По умолчанию использует виджет TextInput.
   - EmailField. Тот же CharField, но проверяет, что введен валидный адрес электронной почты.
   - URLField. Проверяет, что строка является корректной ссылкой.
   - SlugField. Разрешает только буквы, цифры, дефисы и подчеркивания (обычно для URL-адресов статей).
   - RegexField. Позволяет задать регулярное выражение, которому должен соответствовать текст.
2. Числовые поля
   - IntegerField. Для целых чисел. Проверяет, что введено именно число.
   - DecimalField. Для дробных чисел с фиксированной точностью (удобно для цен). Требует аргументы max_digits (всего цифр) и decimal_places (знаков после запятой).
   - FloatField. Для чисел с плавающей точкой.

Поля формы

3. Поля выбора (Choices)

- ChoiceField. Выпадающий список. Ожидает список кортежей в аргументе choices.
- MultipleChoiceField. Позволяет выбрать несколько вариантов из списка.
- TypedChoiceField. То же самое, что ChoiceField, но умеет приводить выбранное значение к определенному типу (например, к int).

4. Поля даты и времени

- DateField. Для дат.
- DateTimeField. Для даты и времени.
- TimeField. Только для времени.
- DurationField. Для промежутков времени (типа timedelta в Python).

5. Логические поля

- BooleanField. Чекбокс («галочка»). Возвращает True или False.
- NullBooleanField. Позволяет выбрать «Да», «Нет» или «Неизвестно» (использует Select).

6. Загрузка файлов

- FileField. Для загрузки файлов.

Универсальные аргументы полей формы

У всех полей Django-формы (будь то CharField, IntegerField и т.д.) есть набор общих аргументов, которые управляют их поведением, внешним видом и логикой проверки.

required
По умолчанию равен True. Если оставить поле пустым, форма не пройдет валидацию.
required=False делает возможным незаполнение формы

label
Строка, которая будет отображаться рядом с полем в браузере (в теге <label>). По умолчанию Django создает название из имени переменной (например, sender_name станет «Sender name»). Аргумент label="Ваше имя" сделает его человекочитаемым.

initial
Значение, которое будет вписано в поле по умолчанию при первой отрисовке формы.

widget
Класс, который отвечает за HTML-код поля.
Одно и то же поле CharField можно отобразить как обычный <input type="text"> (по умолчанию) или как <textarea>, если указать widget=forms.Textarea

Универсальные аргументы полей формы

help_text
Дополнительная подсказка (инструкция для пользователя), которая выводится под полем.

error_messages
Словарь, позволяющий перезаписать стандартные сообщения об ошибках.
Пример: error_messages={'required': 'Куда же вы без логина?'}

validators
Список дополнительных функций для проверки данных. Если встроенной проверки (например, на email) мало, можно добавить свою логику.

disabled
Если True, поле отображается, но его нельзя редактировать. Используется для отображения данных, которые пользователю запрещено менять

```
name = forms.CharField(
    label="Имя пользователя",
    required=True,
    initial="Аноним",
    help_text="Введите ваше настоящее имя",
    widget=forms.TextInput(attrs={'class': 'my-css-class'})
)
```

Widget

Widget (Виджет) — это «лицо» поля. Если само поле (Field) отвечает за логику (проверку данных, перевод типов), то виджет отвечает исключительно за генерацию HTML-кода и извлечение данных из словаря POST-запроса.

Каждое поле имеет виджет по умолчанию, но его можно изменить:

```
class MyForm(forms.Form):
    # Обычная строка станет многострочным текстовым полем
    comment = forms.CharField(widget=forms.Textarea)

    # Поле для пароля (скрывает символы)
    password = forms.CharField(widget=forms.PasswordInput)

    # Скрытое поле для технических данных
    user_id = forms.IntegerField(widget=forms.HiddenInput)
```

Widget

Аргумент attrs это самая частая причина использования виджетов. Через атрибут attrs передается словарь, который превратится в HTML-атрибуты.

```
name = forms.CharField(
    widget=forms.TextInput(attrs={
        'class': 'form-control',    # Добавляем класс для Bootstrap
        'placeholder': 'Введите имя',
        'id': 'unique_name_id'      # Можно задать свой ID
    })
)
```

Результат в HTML:

```
<input type="text" class="form-control" placeholder="Введите имя" id="unique_name_id">
```

Популярные встроенные виджеты

Поле	Виджет по умолчанию	Альтернативы
CharField	TextInput	Textarea, PasswordInput, HiddenInput,
BooleanField	CheckboxInput	
ChoiceField	Select	RadioSelect
MultipleChoiceField	SelectMultiple	CheckboxSelectMultiple
DateField	DateInput	
FileField	ClearableFileInput	FileInput

Очистка данных формы

В Django очистка данных (Cleaning) — это процесс превращения сырого текста из HTML-формы в чистые объекты Python (числа, даты, списки) и их одновременная проверка.

Процесс запускается методом `is_valid()`. Если проверка прошла успешно, данные попадают в словарь `form.cleaned_data`.

Различают три уровня очистки:

1. **Автоматическая очистка** (уровень поля). Каждое поле само знает, как очистить в нем данные.

`IntegerField` превратит строку "42" в число 42.

`EmailField` выдаст ошибку, если в строке нет символа @.

`DateField` превратит "2023-10-25" в объект `datetime.date`.

2. **Очистка конкретного поля** (реализуется в методе `clean_<fieldname>)`

Если вам нужно добавить специфическую проверку для одного поля (например, запретить регистрацию с определенным логином), создается метод с именем clean_ + название поля.

3. **Валидация всей формы сразу** (реализуется в методе `clean())`

Используется, когда нужно сравнить данные из нескольких полей. Классический пример — проверка совпадения пароля и его подтверждения.

Параметры отображения формы

Параметры отображения в Django можно разделить на два уровня: как форма выводится целиком (быстрые способы) и как стилизовать каждое поле в отдельности (тонкая настройка).

Быстрые способы вывода (стандартные методы)

- **`{{ form.as_p }}`** — каждое поле оборачивается в тег `<p>` (самый популярный вариант).
- **`{{ form.as_table }}`** — выводит поля как строки таблицы `<tr>` (вам нужно самому создать `<table>` вокруг).
- **`{{ form.as_ul }}`** — выводит поля как список `<li>` (нужен тег `<ul>` вокруг).

Параметры отображения формы

Ручное управление полями (Iterating fields)

```
<form method="post">
    {% csrf_token %}
    {% for field in form %}
        <div class="field-wrapper">
            <label for="{{ field.id_for_label }}">{{ field.label }}:</label>
            {{ field }}
            {% if field.help_text %}
                <small class="help">{{ field.help_text }}</small>
            {% endif %}
            {{ field.errors }}
        </div>
    {% endfor %}
    <button type="submit">Отправить</button>
</form>
```

Формы на основе моделей

Если нужно создать форму для создания или редактирования объекта, который уже описан в моделях (models.py), использовать обычный forms.Form — лишняя работа. Для этого в Django есть ModelForm.

ModelForm автоматически делает три вещи:

- Создает поля формы на основе типов полей модели.
- Сама проверяет данные (валидация).
- Сохраняет данные в базу одной командой.

Модели для работы

```
from datetime import date
from django.db import models

  class Blog(models.Model):
     name = models.CharField(max_length=100, verbose_name="Название блога"))
     tagline = models.TextField(verbose_name="Теглайн")

  class Author(models.Model):
     name = models.CharField(max_length=200)
     email = models.EmailField()

  class Entry(models.Model):
     blog = models.ForeignKey(Blog, on_delete=models.CASCADE)
     headline = models.CharField(max_length=255)
     body_text = models.TextField()
     pub_date = models.DateField()
     mod_date = models.DateField(default=date.today)
     authors = models.ManyToManyField(Author)
     number_of_comments = models.IntegerField(default=0)
     number_of_pingbacks = models.IntegerField(default=0)
     rating = models.IntegerField(default=5)
```

Формы на основе моделей

blogs/forms.py

```
from django import forms
from .models import Blog
class BlogForm(forms.ModelForm):
    class Meta:
        model = Blog
        fields = ['name', 'tagline’]
```

Ключевые параметры в классе Meta

- model: Класс модели, на которой основана форма.
- fields: Список полей, которые должны быть в форме. Можно использовать '__all__', чтобы вывести всё сразу (но это считается плохим тоном в плане безопасности).
- exclude: Список полей, которые нужно исключить (наоборот).
- widgets: Словарь для настройки стилей.
- labels / help_texts / error_messages: Позволяют переопределить настройки из модели специально для этой формы.

Class Based Views + form

blogs/view.py

```
from django.shortcuts import render
from django.urls import reverse_lazy
from django.views.generic import ListView
from django.views.generic import CreateView, UpdateView, DeleteView
from .models import Blog
from .forms import BlogForm

class BlogListView(ListView):
    model = Blog
    template_name = 'blogs/blog/blog_list.html'
    context_object_name = 'blogs'  # Имя переменной, которая будет доступна в шаблоне

# Создание
class BlogCreateView(CreateView):
    model = Blog
    form_class = BlogForm
    template_name = 'blogs/blog/blog_form.html'
    success_url = reverse_lazy('blog_list') # Куда редиректить после успеха
```

Class Based Views + form

```
# Обновление
class BlogUpdateView(UpdateView):
    model = Blog
    form_class = BlogForm
    template_name = 'blogs/blog/blog_form.html' # Можно использовать тот же
 шаблон
    success_url = reverse_lazy('blog_list')


# Удаление
class BlogDeleteView(DeleteView):
    model = Blog
    template_name = 'blogs/blog/blog_confirm_delete.html'
    success_url = reverse_lazy('blog_list')
```

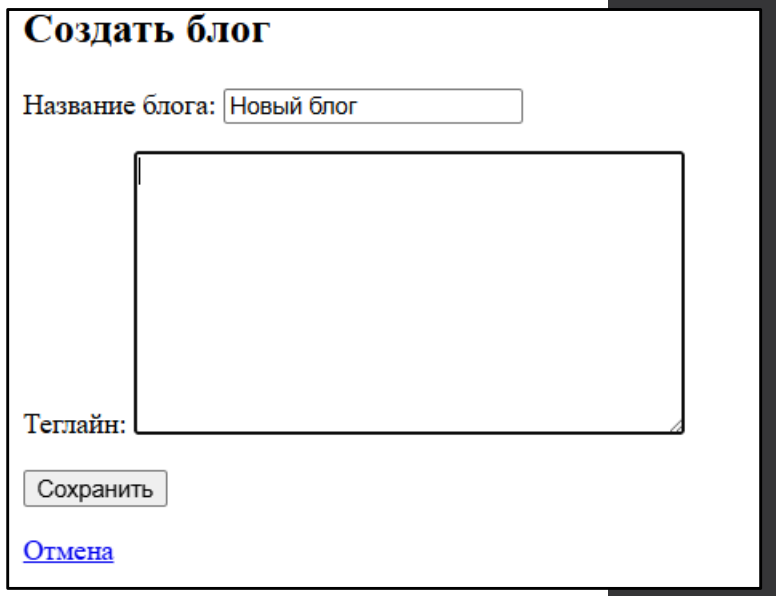
Шаблоны

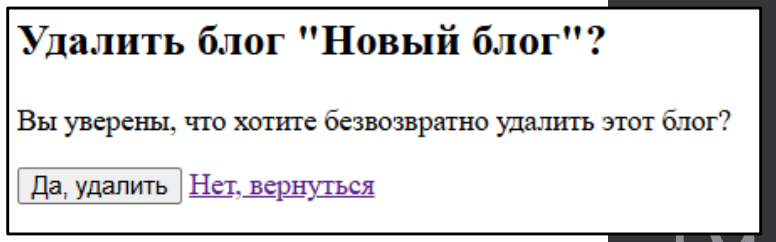
blogs/templates/blogs/blog/blog_form.html

```
<h2>{% if object %}Редактировать{% else %}Создать{% endif %} блог</h2>
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Сохранить</button>
</form>
<a href="{% url 'blog_list' %}">Отмена</a>
```

blogs/templates/blogs/blog/blog_confirm_delete.html

```
<h2>Удалить блог "{{ object.name }}"?</h2>
<form method="post">
    {% csrf_token %}
    <p>Вы уверены, что хотите безвозвратно удалить этот блог?</p>
    <button type="submit">Да, удалить</button>
    <a href="{% url 'blog_list' %}">Нет, вернуться</a>
</form>
```





Маршрутизация

blogs/urls.py

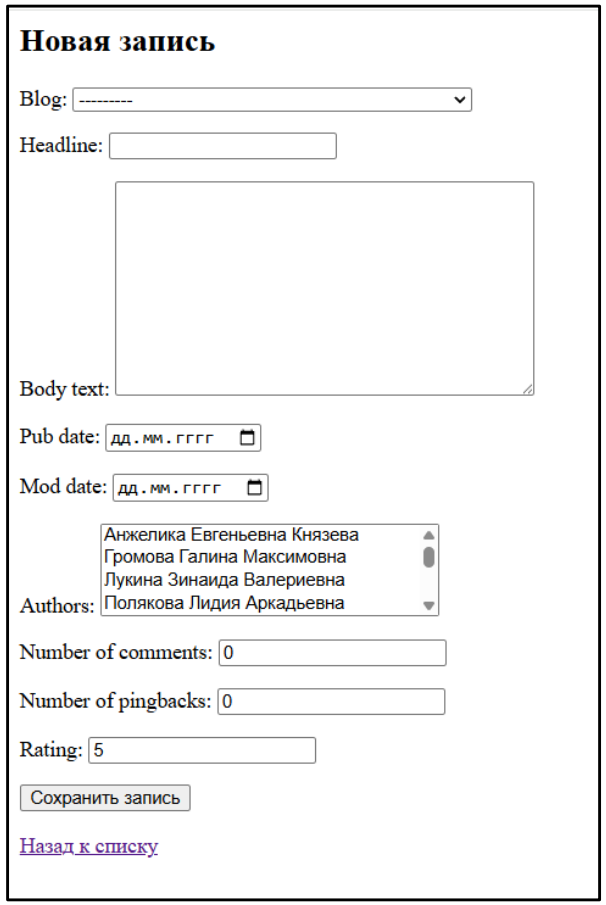
```
from django.urls import path
from . import views

urlpatterns = [
    path('blog/add/', views.BlogCreateView.as_view(), name='blog_add'),
    path('blog/<int:pk>/edit/', views.BlogUpdateView.as_view(), name='blog_edit'),
    path('blog/<int:pk>/delete/', views.BlogDeleteView.as_view(), name='blog_delete'),
    path('blogs/', views.BlogListView.as_view(), name='blog_list'),
]
```

Форма для более сложной модели Entry

blogs/forms.py:

```
class EntryForm(forms.ModelForm):
    class Meta:
        model = Entry
        fields = '__all__’
        # Или перечислите нужные:
        # ['blog', 'headline', 'body_text', 'pub_date', 'authors']
        widgets = {
            'pub_date': forms.DateInput(attrs={'type': 'date'}),
            'mod_date': forms.DateInput(attrs={'type': 'date'}),
        }
```

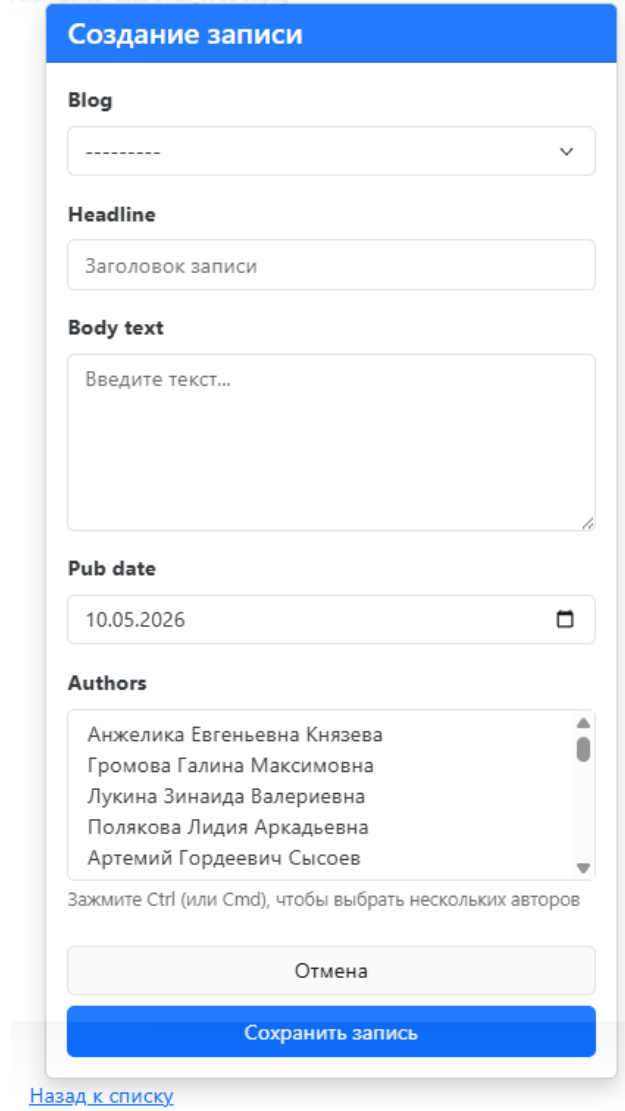
В ModelForm виджеты можно переопределить в специальном словаре внутри класса Meta: `widgets`



Форма для более сложной модели Entry

blogs/forms.py:

```
from .models import Entry
class EntryForm(forms.ModelForm):
    class Meta:
        model = Entry
        fields = ['blog', 'headline', 'body_text', 'pub_date', 'authors']
        widgets = {
            'blog': forms.Select(attrs={'class': 'form-select'}),
            'headline': forms.TextInput(attrs={
                'class': 'form-control',
                'placeholder': 'Заголовок записи'
            }),
            'body_text': forms.Textarea(attrs={
                'class': 'form-control',
                'rows': 5,
                'placeholder': 'Введите текст...'
            }),
            'pub_date': forms.DateInput(attrs={
                'class': 'form-control',
                'type': 'date'
            }),
            'authors': forms.SelectMultiple(attrs={
                'class': 'form-select',
                'size': '5'
            }),
        }

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.fields['authors'].help_text = "Зажмите Ctrl (или Cmd), чтобы выбрать нескольких авторов"
```



Шаблон для Entry

blogs/templates/entry/entry_form.html

```
<div class="container mt-5">
    <div class="card shadow">
        <div class="card-header bg-primary text-white">
            <h4 class="mb-0">{% if object %}Редактирование{% else %}Создание{% endif %} записи</h4>
        </div>
        <div class="card-body">
            <form method="post" novalidate>
                {% csrf_token %}
                {% for field in form %}
                    <div class="mb-3">
                        <label for="{{ field.id_for_label }}" class="form-label fw-bold">
                            {{ field.label }}
                        </label>
                        {{ field }}
                        {% if field.help_text %}
                            <div class="form-text text-muted">{{ field.help_text }}</div>
                        {% endif %}
                        {% if field.errors %}
                            {% for error in field.errors %}
                                <div class="text-danger small mt-1">{{ error }}</div>
                            {% endfor %}
                        {% endif %}
                    </div>
                {% endfor %}
                <div class="d-grid gap-2 d-md-flex justify-content-md-end mt-4">
                    <a href="{% url 'entry_list' %}" class="btn btn-light border">Отмена</a>
                    <button type="submit" class="btn btn-primary">Сохранить запись</button>
                </div>
            </form>
        </div>
    </div>
</div>
<a href="{% url 'entry_list' %}">Назад к списку</a>
```