Разработка программного обеспечения на языке Python

Лекция 3. Шаблонизация в django

Посмотрите видеоуроки и ответьте на контрольные вопросы после лекции

Настройка форм в Django

В формах Django мы можем использовать различные классы для создания полей форм. Рассмотрим некоторые из них.

- **BooleanField**: создает поле типа checkbox.
- **CharField**: предназначен для ввода текста.
- **EmailField**: предназначен для ввода адреса электронной почты.
- **FileField()**, **ImageField()**: предназначен для выбора файла.
- **DateField()**: предназначен установки даты.
- **TimeField()**: для ввода времени,
- **IntegerField()**, **DecimalField()**, **FloatField()**: предназначен для ввода чисел.
- **ChoiceField(choises=кортеж_кортежей)**: генерирует список select.

При создании форм для их полей может быть задан **набор свойств** для их настройки.

Свойство **label** позволяет установить текстовую метку, которая отображается рядом с полей. По умолчанию она отображает название самого поля с большой буквы

Параметр **widget** позволяет задать виджет (тип поля формы), который будет использоваться для генерации разметки html.

С помощью параметра **initial** можно установить значения по умолчанию. Поля ввода отображаются на веб-странице в том порядке, в котором они определены в классе формы.

С помощью свойства **field_order** можно переопределить порядок.

Далее приведем пример добавления настройки параметров в класс формы.

forms.py

```python
from django import forms


class UserForm(forms.Form):
    # label - текст перед полем формы
    name = forms.CharField(label="Имя")
    # widget - тип поля формы, initial - начальное значние поля
    age = forms.IntegerField(label="Возраст", widget=forms.Textarea, initial=18)
    # Порядок полей формы
    field_order = ["age", "name"]
```

Валидация данных. Теоретически пользователь может ввести в форму какие угодно данные и отправить их. Однако не все данные бывают уместными или корректными. Например, в поле для возраста пользователь может ввести отрицательное число. В этой связи для проверки корректности вводимых данных используется механизм валидации.

Основным элементом валидации являются правила, которые задают параметры корректности вводимых данных.

Например, для всех полей по умолчанию устанавливается обязательность ввода значения. И если мы попробуем отправить форму, если какое-то из ее полей не введено никакого значения, то мы получим ошибку. Допустим, одно поле может иметь, а может не иметь значение. В этом случае мы можем отключить атрибут **required.** Для полей, которые требуют ввода текста, можно задать соответственно максимальную и минимальную длину вводимого текста в символах. Для числовых объектов можно задать максимально и минимально допустимое значение.

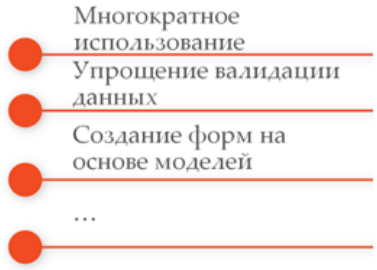
is_valid. Выше рассмотренные атрибуты позволяют валидировать значения при вводе на стороне клиента. Однако практически пользователя, имея определенные навыки, могут все равно отправить форму с заведомо некорректными данными. Поэтому проверку на валидность данных также надо определять на стороне сервера. Для этого у формы вызывается метод **is_valid()**, который возвращает True, если данные корректны, и False - если данные некорректны. Чтобы использоать этот метод, надо создать объект формы и передать ей пришедшие из запроса данные.

views.py

```python
from django.shortcuts import render
from django.http import HttpResponse
from .forms import UserForm


def index(request):
    if request.method == "POST":
        userform = UserForm(request.POST)
        # проверка валидности данных
        if userform.is_valid():
            # получение данных поля name формы
            name = userform.cleaned_data["name"]
            return HttpResponse(f"<h2>Hello, {name}</h2>")
        else:
            return HttpResponse("Invalid data")
    else:
        userform = UserForm()
        return render(request, "index.html", {"form": userform})
```

Итак, определим следующее представление в файле **views.py**. Если приходит POST-запрос, то в начале заполняем форму пришедшими данными. Потом проверяем их корректность. После проверки на валидность мы можем получить данные через объект cleaned_data (если данные корректны). Если данные некорректны, можно предусмотреть альтернативный вывод.



Таким образом, набор инструментов django позволяет определять функциональность форм в одном месте и использовать многократно в разных местах, упрощают валидацию данных, помогают связывать формы с моделями и многое другое. В этой теме были представлены сведения о формах. Мы узнали, как определить и настроить в форме поля, как проверить корректность введенных в форму данных. Теперь нам нужно разобраться с тем, как Django работает с самими данными. То есть понять, как можно сохранить в БД данные, которые пользователь ввел в форме, или как получить из БД данные, запрошенные пользователем. В Django все действия с информацией, которая хранится в БД, осуществляются через модели.

Вопросы

ПРЕДЫДУЩИЙ ЭЛЕМЕНТ КУРСА

◄ Задание 3. Создание своего первого сайта на Django

Перейти на...

СЛЕДУЮЩИЙ ЭЛЕМЕНТ КУРСА

Задание 4. Создание страниц веб-приложения ►