Конспект: Основи шаблонів Django

1. Знайомство з шаблонами

Що таке шаблони?

Шаблони (templates) в Django — це файли, які відповідають за зовнішній вигляд вашого веб-застосунку. Вони дозволяють відокремити дизайн (HTML-код) від бізнес-логіки (Python-коду), що робить проєкт більш організованим і легшим для підтримки.

Шаблони використовують спеціальний синтаксис, який дозволяє впроваджувати в HTML-код дані, що надсилаються з вашого Python-коду. Цей синтаксис називається **Мова шаблонів Django (Django Template Language)**.

Налаштування шаблонів у settings.py

Щоб Django знав, де шукати ваші файли-шаблони, він використовує змінну TEMPLATES у файлі settings.py.

Ось як виглядає конфігурація за замовчуванням:

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Давайте розберемо кожен параметр, щоб все було зрозуміло:

- **BACKEND**: Це рушій шаблонів, який Django буде використовувати. За замовчуванням це DjangoTemplates.
- **DIRS**: Це список шляхів до тек. Зазвичай тут вказують загальні теки з шаблонами, які

можуть бути спільними для всього проєкту. За замовчуванням він порожній, і для невеликих проєктів так і залишається.
- **APP_DIRS**: Цей параметр має значення True за замовчуванням. Це означає, що Django буде автоматично шукати теку з назвою templates всередині кожного вашого застосунку. Цей спосіб є найзручнішим і найчастіше використовується.

Структура проєкту з шаблонами

Як тільки ви створили свій застосунок (наприклад, hello), ви можете додати в нього теку templates. Для уникнення конфліктів назв файлів рекомендується створити всередині templates ще одну теку з назвою вашого застосунку.

- **Приклад структури проєкту:**
  myproject/
  ├── hello/
  │ ├── templates/
  │ │ └── hello/
  │ │ └── index.html # Ваш шаблон знаходиться тут
  │ ├── views.py
  ├── myproject/
  │ ├── settings.py
  └── manage.py

Далі, у файлі index.html ви можете написати звичайний HTML-код.

```
<!DOCTYPE html>
<html>
<head>
    <title>Django на HELLO.COM</title>
    <meta charset="utf-8" />
</head>
<body>
    <h2>Hello HELLO.COM</h2>
</body>
</html>
```

2. Зв'язок шаблону та представлення (View)

Використання шаблону у представленні

Щоб показати користувачеві сторінку, нам потрібно "з'єднати" файл шаблону з функцією-представленням (view). Для цього ми використовуємо функцію render(), яка

імпортується з django.shortcuts.

- **Приклад у views.py:**

```
from django.shortcuts import render

def index(request):
    # Функція render() приймає:
    # 1. Об'єкт запиту (request)
    # 2. Шлях до шаблону
    # 3. Необов'язковий словник з даними (context)
    return render(request, "hello/index.html")
```

Зверніть увагу: Шлях до шаблону "hello/index.html" вказує Django, що потрібно шукати файл index.html всередині теки hello, яка знаходиться в теці templates.

Налаштування маршруту (URL)

Після того, як ви створили представлення, його потрібно "підв'язати" до певного URL-адреси. Це робиться у файлі urls.py.

- **Приклад у urls.py:**

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

urlpatterns = [
    # Зіставляємо кореневий URL ("") з функцією index
    path("", views.index),
]
```

Тепер, коли ви запустите сервер і відкриєте браузер за адресою http://127.0.0.1:8000/, Django викличе функцію index, яка відрендерить та покаже вам ваш HTML-шаблон.

Додавання інших шаблонів

Процес додавання нових сторінок точно такий самий.

1. **Створюємо HTML-файли:** Додаємо about.html та contact.html до теки templates/hello.
2. **Пишемо функції у views.py:**

```python
from django.shortcuts import render

def index(request):
    return render(request, "hello/index.html")

def about(request):
    return render(request, "hello/about.html")

def contact(request):
    return render(request, "hello/contact.html")
```

3. **Додаємо маршрути в urls.py:**

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

urlpatterns = [
    path("", views.index),
    path("about/", views.about),
    path("contact/", views.contact),
]
```

Використання TemplateResponse

Для генерації шаблону найчастіше використовується render(). Проте, ви можете зустріти й інший спосіб — клас TemplateResponse.

- **Приклад у views.py:**

```python
from django.template.response import TemplateResponse

def index(request):
    return TemplateResponse(request, "hello/index.html")
```

Результат для користувача буде такий самий. Головна відмінність полягає в тому, що

TemplateResponse дозволяє відкласти сам процес рендерингу (перетворення шаблону в HTML) до моменту, коли це дійсно потрібно, що може бути корисно для складних систем.

3. Передача даних у шаблони

Передача простих даних

Одна з найважливіших можливостей шаблонів — це передача динамічних даних з представлення. Дані передаються у вигляді словника, який називається **context**, і є третім параметром функції render().

- **Приклад у views.py:**

```
from django.shortcuts import render

def index(request):
    # Створюємо словник з даними, які хочемо передати
    data = {
        "header": "Привіт, Django!",
        "message": "Ласкаво просимо до світу Python."
    }
    # Передаємо словник через параметр context
    return render(request, "hello/index.html", context=data)
```

- **Приклад у index.html:**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Django на HELLO.COM</title>
</head>
<body>
    <!-- Використовуємо змінні, які прийшли з представлення -->
    <h2>{{ header }}</h2>
    <p>{{ message }}</p>
</body>
</html>
```

Все, що вам потрібно зробити, це взяти ключ зі словника (header, message) і обгорнути його в подвійні фігурні дужки {{ }}. Django автоматично замінить ці змінні на їхні значення.

Передача складних даних

Шаблони Django легко працюють зі складними типами даних, такими як словники, списки та кортежі. Доступ до їхніх елементів здійснюється через крапку (.) та індекси.

- **Приклад у views.py:**

```
from django.shortcuts import render

def complex_data_view(request):
    header = "Дані користувача"
    langs = ["Python", "Java", "C#"]
    user = {"name": "Tom", "age": 23}
    address = ("Шевченко", 23, 45)

    data = {
        "header": header,
        "langs": langs,
        "user": user,
        "address": address
    }
    return render(request, "hello/complex_data.html", context=data)
```

- **Приклад у complex_data.html:**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Django на HELLO.COM</title>
</head>
<body>
    <h1>{{ header }}</h1>

    <h3>Інформація про користувача</h3>
    <p>Ім'я: {{ user.name }}</p>
    <p>Вік: {{ user.age }}</p>
```

```
    <h3>Адреса</h3>
    <p>Вулиця: {{ address.0 }}</p>
    <p>Будинок: {{ address.1 }}</p>
    <p>Квартира: {{ address.2 }}</p>

    <h3>Улюблені мови програмування</h3>
    <p>Перша: {{ langs.0 }}</p>
    <p>Друга: {{ langs.1 }}</p>
</body>
</html>
```

- **Доступ до словника:** {{ user.name }}
- **Доступ до кортежу/списку:** {{ address.0 }}

Передача об'єктів класів

Ви можете передавати в шаблони об'єкти ваших власних класів. Доступ до їхніх атрибутів також здійснюється через крапку.

- **Приклад у views.py:**
  from django.shortcuts import render

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

def person_view(request):
    # Створюємо об'єкт класу Person
    person_obj = Person("Tom", 30)
    # Передаємо об'єкт у шаблон
    return render(request, "hello/person.html", context={"person":
person_obj})
```

- **Приклад у person.html:**

```
<!DOCTYPE html>
<html>
```

```
<head>
    <meta charset="utf-8" />
    <title>Django на HELLO.COM</title>
</head>
<body>
    <!-- Звертаємося до атрибутів об'єкта через крапку -->
    <h1>Інформація про особу:</h1>
    <p>Ім'я: {{ person.name }}</p>
    <p>Вік: {{ person.age }}</p>
</body>
</html>
```

Вбудовані теги шаблонів Django

Що таке теги шаблонів?

Теги шаблонів — це спеціальні конструкції в Мові шаблонів Django, які дозволяють додавати логіку до вашого HTML-коду. Вони обробляються рушієм шаблонів і допомагають динамічно змінювати контент, наприклад, відображати списки, керувати умовним виведенням та інше. Теги завжди знаходяться між символами {% ... %}.

1. Тег autoescape

Що робить autoescape?

Тег autoescape контролює, чи буде Django автоматично екранувати HTML-символи у ваших даних. Екранування — це процес заміни спеціальних символів HTML (<, >, &, ', ") на їхні безпечні еквіваленти (<, >, &, ', ").

Це дуже важлива функція безпеки, яка допомагає запобігти **XSS-атакам (Cross-Site Scripting)**. За замовчуванням autoescape увімкнений, що є найкращою практикою.

- **Приклад у views.py:**

```
from django.shortcuts import render

def index(request):
    # Цей рядок містить HTML-тег
    data = {"body": "<h1>Привіт Світ!</h1>"}
    return render(request, "hello/index.html", context=data)
```

- **Приклад у index.html:**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Django на HELLO.COM</title>
</head>
<body>
    <p><b>Вивід без відключення autoescape:</b></p>
    {{ body }}

    <p><b>Вивід з відключеним autoescape:</b></p>
    {% autoescape off %}
        {{ body }}
    {% endautoescape %}
</body>
</html>
```

Пояснення:

- **Перший вивід** ({{ body }}): Оскільки autoescape увімкнений за замовчуванням, тег <h1> буде екранований. У браузері ви побачите просто текст <h1>Привіт Світ!</h1>.
- **Другий вивід** ({% autoescape off %}): Тут ми примусово відключаємо екранування. Браузер інтерпретує рядок як справжній HTML, і ви побачите заголовок Привіт Світ!, відформатований як h1.

2. Коментарі

Як додавати коментарі в шаблони?

Коментарі — це нотатки для розробників, які ігноруються Django. Вони допомагають пояснити складні ділянки коду.

- **Однорядковий коментар:** Використовуйте {# ... #}.
  {# Це простий однорядковий коментар #}

- **Блоковий коментар:** Використовуйте тег {% comment %} для багаторядкових коментарів.

```
{% comment %}
    Це великий блок коментарів,
    який може охоплювати кілька рядків.
    Він ігнорується повністю.
{% endcomment %}
<p>Тільки цей текст буде видимим.</p>
```

3. Умовні вирази if...else

Як використовувати if...else?

Теги if дозволяють виконувати логічні перевірки і відображати певний контент лише тоді, коли умова виконується.

- **Базовий if:**

```
{% if умова %}
    <!-- Цей контент буде показано, якщо умова істинна -->
{% endif %}
```

- **Приклад у views.py:**

```
from django.shortcuts import render

def index(request):
    data = {"number": 5}
    return render(request, "hello/index.html", context=data)
```

- **Приклад у index.html:**

```
{% if number > 0 %}
    <p>Число позитивне.</p>
{% endif %}
```

Оскільки number (5) більше за 0, текст буде відображено.

Додаткові теги else та elif

- **Тег else**: Дозволяє вивести контент, якщо умова в if хибна.

```
{% if number > 0 %}
    <p>Число позитивне.</p>
{% else %}
    <p>Число негативне або дорівнює нулю.</p>
{% endif %}
```

- **Тег elif**: Дозволяє перевірити додаткові умови, якщо попередня була хибною.

```
{% if number > 0 %}
    <p>Число позитивне.</p>
{% elif number < 0 %}
    <p>Число негативне.</p>
{% else %}
    <p>Число дорівнює нулю.</p>
{% endif %}
```

Логічні оператори в if

Ви можете використовувати логічні оператори для створення складніших умов: and, or, not.

- **Приклад:**

```
{% if a > 0 and b > 0 %}
    <p>Обидва числа позитивні.</p>
{% elif a > 0 or b > 0 %}
    <p>Хоча б одне число позитивне.</p>
{% else %}
    <p>Жодне число не є позитивним.</p>
{% endif %}
```

4. Цикли for

Як використовувати for?

Тег for дозволяє перебирати елементи колекцій, таких як списки або словники.

- **Базовий синтаксис:**

```
{% for елемент in колекція %}
    <!-- дії з елементом -->
{% endfor %}
```

- **Приклад у views.py:**

```
from django.shortcuts import render

def index(request):
    langs = ["Python", "JavaScript", "Java", "C#"]
    return render(request, "hello/index.html", context={"langs": langs})
```

- **Приклад у index.html:**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Django на HELLO.COM</title>
</head>
<body>
    <h2>Мови програмування</h2>
    <ul>
        {% for lang in langs %}
            <li>{{ lang }}</li>
        {% endfor %}
    </ul>
</body>
```

```
</html>
```

Цей код автоматично створить елемент списку <li> для кожного рядка у списку langs.

Тег empty

Що робити, якщо список порожній? Для цього існує тег {% empty %}.

```
<ul>
    {% for lang in langs %}
        <li>{{ lang }}</li>
    {% empty %}
        <li>Наразі список мов порожній.</li>
    {% endfor %}
</ul>
```

Якщо список langs буде порожнім, замість циклу відобразиться текст всередині {% empty %}.

Ітерація по словниках

Ви також можете перебирати словники за допомогою циклу for. Для цього використовуйте метод .items() як у Python.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Django на HELLO.COM</title>
</head>
<body>
    {% for key, value in colors.items %}
        <p>{{ key }}: {{ value }}</p>
    {% endfor %}
</body>
</html>
```

- **Приклад у views.py:**

```python
from django.shortcuts import render

def index(request):
    colors = {"red": "червоний", "green": "зелений", "blue": "синій"}
    return render(request, "hello/index.html", context={"colors": colors})
```

5. Тег with

Як використовувати with?

Тег with дозволяє створити тимчасову змінну, яку можна використовувати лише в межах певного блоку. Це корисно для спрощення складних виразів.

```
{% with name="Tom" age=29 %}
    <div>
        <p>Ім'я: {{ name }}</p>
        <p>Вік: {{ age }}</p>
    </div>
{% endwith %}
<!-- Тут змінні `name` та `age` вже недоступні! -->
```

Ці змінні існують лише всередині блоку {% with %}...{% endwith %}.

6. Тег now

Як використовувати now?

Тег now відображає поточний системний час. Він приймає рядок форматування, щоб вивести дату та час у потрібному форматі.

- **Приклад у index.html:**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Django на HELLO.COM</title>
```

```
</head>
<body>
    <p>Поточний рік: {% now "Y" %}</p>
    <p>Повна дата: {% now "F j, Y" %}</p>
    <p>Дата та час: {% now "N j, Y, P" %}</p>
</body>
</html>
```

- **Основні символи форматування:**
    - Y: Рік з чотирма цифрами (наприклад, 2025)
    - F: Повна назва місяця (наприклад, Серпень)
    - j: День місяця без початкового нуля (1-31)
    - N: Скорочена назва місяця (наприклад, Сер)
    - P: Час у форматі AM/PM (наприклад, 4 p.m.)

Зміна часового поясу

Django використовує часовий пояс, вказаний у файлі settings.py у змінній TIME_ZONE. За замовчуванням це UTC. Ви можете змінити його на свій, наприклад:

```
TIME_ZONE = 'Europe/Kyiv'
```

Фільтри шаблонів Django

Що таке фільтри шаблонів?

Фільтри — це спеціальні функції, які дозволяють змінювати, форматувати або обробляти дані, що виводяться в шаблоні. На відміну від тегів, які починаються з {% ... %}, фільтри позначаються символом вертикальної риски |.

Загальний синтаксис: {{ змінна | назва_фільтра }}.

1. Фільтр add

Що робить add?

Фільтр add додає одне значення до іншого. Він може працювати як з числами (виконуючи додавання), так і з рядками (виконуючи конкатенацію).

Синтаксис: {{ змінна | add:значення_для_додавання }}.

- **Приклад з рядками:**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Django | Фільтр add</title>
</head>
<body>
    {% with message="Hello" %}
        <h2>{{ message|add:" Django" }}</h2>
    {% endwith %}
</body>
</html>
```

Результат: <h2>Hello Django</h2>

- **Приклад з числами:**

```
{% with x=10 y=5 %}
    <p>Сума: {{ x|add:y }}</p>
{% endwith %}
```

Результат: Сума: 15

2. Фільтр capfirst

Що робить capfirst?

Фільтр capfirst робить першу букву рядка великою. Це корисно для форматування заголовків або імен.

- **Приклад:**

```
{% with name="django framework" %}
    <p>{{ name|capfirst }}</p>
{% endwith %}
```

Результат: Django framework

3. Фільтр cut

Що робить cut?

Фільтр cut видаляє з рядка всі входження заданої підстроки. Це зручно, якщо вам потрібно прибрати певні символи або слова.

- **Приклад:**

```
{% with sentence="Я був вдома" %}
    <p>{{ sentence|cut:"був" }}</p>
{% endwith %}
```

Результат: Я вдома

4. Перевірка значення та значення за замовчуванням

Фільтр default

Фільтр default перевіряє значення. Якщо воно "хибне" (тобто False, None, порожній рядок "", порожній список [] тощо), фільтр повертає вказане вами значення за замовчуванням.

- **Приклад з порожнім рядком:**

```
{% with name="" %}
    <p>Ім'я: {{ name|default:"Не визначено" }}</p>
{% endwith %}
```

Результат: Ім'я: Не визначено

Фільтр default_if_none

Фільтр default_if_none працює схоже, але перевіряє значення лише на None. Якщо значення None, він повертає значення за замовчуванням.

- **Приклад:**

```
{% with user=None %}
    <p>Користувач: {{ user|default_if_none:"Анонімний користувач" }}</p>
{% endwith %}
```

Результат: Користувач: Анонімний користувач

5. Фільтр floatformat

Що робить floatformat?

Фільтр floatformat форматує числа з плаваючою точкою, округлюючи їх до заданої кількості знаків після коми.

- **Базове округлення (без аргументів):** Округлює до одного знака після коми.

```
<p>{{ 2.56|floatformat }}</p>
```

Результат: 2.6

- **Округлення до N знаків:** Передайте число як аргумент.

```
<p>{{ 2.5678|floatformat:2 }}</p>
```

Результат: 2.57

- **Округлення з відсіканням нулів:** Використовуйте від'ємне число. Нулі після коми будуть відсічені.

```
<p>{{ 2.500|floatformat:"-2" }}</p>
<p>{{ 2.560|floatformat:"-2" }}</p>
```

Результат:
2.5
2.56

- **Округлення до найближчого цілого:** Використовуйте 0.

```
<p>{{ 2.56|floatformat:0 }}</p>
```

Результат: 3

Локалізація та групування

Django використовує налаштування локалі, вказані у файлі settings.py (LANGUAGE_CODE). За замовчуванням це en-us. Якщо ви зміните його на uk або uk-ua, то Django буде використовувати українські правила форматування.

- **Приклад у settings.py:**
  `LANGUAGE_CODE = 'uk'`

- **Групування розрядів (g):**

```
{% with value=34232.34 %}
    <p>{{ value|floatformat:"2g" }}</p>
{% endwith %}
```

Результат (з локаллю uk): 34 232,34 . Зверніть увагу на пробіл як розділювач тисяч.

- **Відключення локалізації (u):**

```
{{ value|floatformat:"2u" }}
```

Цей формат відключає локалізацію, і число завжди буде виглядати як 34232.34 незалежно від налаштувань.

6. Форматування дат

Фільтр date

Фільтр date форматує об'єкт datetime у потрібний вам вигляд. Він приймає рядок форматування, що складається зі спеціальних символів.

- **Приклад у views.py:**

```
from datetime import datetime
```

```
from django.shortcuts import render

def index(request):
    return render(request, "hello/index.html", context={"my_date":
datetime.now()})
```

- **Приклад у index.html:**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Django | Форматування дат</title>
</head>
<body>
    <div>
        <h2>Сьогоднішня дата: {{ my_date|date:"d.m.Y" }}</h2>
        <h2>Поточний час: {{ my_date|date:"H:i" }}</h2>
        <h2>ISO 8601: {{ my_date|date:"c" }}</h2>
        <h2>Короткий формат: {{ my_date|date:"SHORT_DATE_FORMAT" }}</h2>
    </div>
</body>
</html>
```

Ключові символи форматування:

- **Дата:** d (день), m (місяць), Y (рік), j (день без нуля), l (повна назва дня тижня).
- **Час:** H (година 24h), i (хвилини), s (секунди), P (формат p.m./a.m.).

7. Операції зі списками

Фільтр join

Фільтр join об'єднує елементи списку в один рядок, використовуючи заданий розділювач.

- **Приклад:**

```
{% with users=["Tom", "Sam", "Bob", "Mike"] %}
    <p>Користувачі: {{ users|join:", " }}</p>
```

```
{% endwith %}
```

Результат: Користувачі: Tom, Sam, Bob, Mike

Фільтр slice

Фільтр slice дозволяє отримати частину списку. Працює як зрізи в Python.

- **Синтаксис:** {{ список|slice:"початок:кінець" }}
- **Приклад:**

```
{% with users=["Tom", "Sam", "Bob", "Mike"] %}
    <p>Частина списку: {{ users|slice:"1:3"|join:", " }}</p>
{% endwith %}
```

Результат: Частина списку: Sam, Bob .

Фільтр length

Фільтр length повертає кількість елементів у списку або довжину рядка.

- **Приклад:**

```
{% with users=["Tom", "Sam", "Bob", "Mike"] %}
    <p>Кількість користувачів: {{ users|length }}</p>
{% endwith %}
```

Статичні файли та розширення шаблонів

1. Статичні файли

Що таке статичні файли?

Статичні файли — це файли, які не змінюються, коли веб-сторінка завантажується. До них належать файли стилів (CSS), скрипти (JavaScript), зображення, шрифти тощо.

Налаштування статичних файлів

Django вже має вбудовану підтримку статичних файлів завдяки застосунку django.contrib.staticfiles, який за замовчуванням увімкнений у файлі settings.py.

- **settings.py:**

```
# URL, який використовується для обслуговування статичних файлів
STATIC_URL = 'static/'

# Список встановлених додатків
INSTALLED_APPS = [
    # ...
    'django.contrib.staticfiles',
    # ...
]
```

Ці налаштування означають, що Django буде шукати теку static/ всередині кожного вашого застосунку.

Структура тек для статичних файлів

Щоб зберігати статичні файли, створіть теку static всередині вашого застосунку. Для кращої організації всередині static можна створити підтеки для різних типів файлів, наприклад:

```
myproject/
├── myapp/
│   ├── static/
│   │   ├── css/
│   │   │   └── styles.css
│   │   ├── images/
│   │   │   └── forest.jpg
│   │   └── js/
│   │       └── script.js
│   ├── templates/
│   │   └── myapp/
│   │       └── index.html
└── manage.py
```

Використання статичних файлів у шаблоні

Щоб підключити статичний файл у вашому HTML-шаблоні, потрібно виконати два кроки:

1. Завантажити тег static на початку шаблону.
2. Використати тег {% static ... %} для отримання правильного шляху до файлу.

- **Приклад у index.html:**

```
<!DOCTYPE html>
{% load static %}
<html>
<head>
    <meta charset="utf-8" />
    <title>Django | Статичні файли</title>
    <!-- Підключаємо CSS-файл -->
    <link rel="stylesheet" href="{% static 'css/styles.css' %}" />
</head>
<body>
    <h1>Зимовий ліс</h1>
    <!-- Додаємо зображення -->
    <img src="{% static 'images/forest.jpg' %}" alt="зимовий ліс" />
</body>
</html>
```

Цей код автоматично перетвориться на шляхи, зрозумілі браузеру, наприклад, /static/css/styles.css.

Налаштування додаткових шляхів

Якщо ви хочете зберігати статичні файли в іншому місці (наприклад, у спільній теці для всього проєкту), ви можете вказати додаткові шляхи в settings.py за допомогою змінної STATICFILES_DIRS.

- **Приклад у settings.py:**

```
STATICFILES_DIRS = [
    BASE_DIR / "static", # Шлях до теки static/ в корені проєкту
    "/var/www/my-static-files/", # Додатковий шлях, якщо потрібно
]
```

2. Шаблони-представлення TemplateView

Що таке TemplateView?

TemplateView — це вбудований клас Django, який дозволяє відрендерити шаблон без написання окремої функції-представлення. Це дуже зручно для простих сторінок, які не потребують складної логіки.

Використання TemplateView

- **Приклад у urls.py:**

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

urlpatterns = [
    path("about/", TemplateView.as_view(template_name="about.html")),
    path("contact/", TemplateView.as_view(template_name="contact.html")),
]
```

Метод as_view() перетворює клас TemplateView на функцію-представлення. Параметр template_name вказує, який шаблон потрібно відрендерити.

Передача даних у шаблон

Ви можете передати дані в шаблон, використовуючи параметр extra_context.

- **Приклад у urls.py:**

```
path("about/", TemplateView.as_view(
    template_name="about.html",
    extra_context={"header": "Про сайт"}
)),
```

У about.html ви можете отримати ці дані:

```
<h1>{{ header }}</h1>
```

3. Розширення шаблонів та тег extends

Навіщо потрібно розширювати шаблони?

Розширення шаблонів — це потужна функція, яка дозволяє створити базовий макет сторінки (наприклад, з хедером, футером та навігацією) і потім повторно використовувати його в інших шаблонах. Це допомагає уникнути дублювання коду і полегшує підтримку сайту.

Тег block

За допомогою тегу {% block назва %} ви визначаєте місця, які можуть бути змінені.

- **Приклад базового шаблону base.html:**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>{% block title %}Базова назва{% endblock title %}</title>
</head>
<body>
    <header>
        <a href="/">Головна</a> | <a href="/contacts">Контакти</a>
    </header>

    <main>
        <h1>{% block header %}{% endblock header %}</h1>
        <div>{% block content %}{% endblock content %}</div>
    </main>

    <footer>
        <p>MyCorp. 2024. Всі права захищені.</p>
    </footer>
</body>
</html>
```

Тут ми визначили три блоки: title, header і content. Блок title має вміст за замовчуванням, який буде використано, якщо дочірній шаблон його не перевизначить.

Тег extends

Дочірні шаблони використовують тег {% extends "шлях_до_базового" %} для наслідування. Потім вони перевизначають вміст блоків, які їм потрібні.

- **Приклад дочірнього шаблону index.html:**

```
{% extends "base.html" %}

{% block title %}Головна{% endblock title %}

{% block header %}Ласкаво просимо!{% endblock header %}
```

- **Приклад іншого шаблону contacts.html:**

```
{% extends "base.html" %}

{% block title %}Контакти{% endblock title %}

{% block header %}Зв'яжіться з нами{% endblock header %}

{% block content %}
<p>Телефон: +12345677890</p>
<p>Email: contact@mycorp.com</p>
{% endblock content %}
```

Вкладені шаблони та фільтр include

Навіщо використовувати вкладені шаблони?

Вкладені шаблони (nested templates) — це шаблони, які можна підключати всередину інших шаблонів. Це дозволяє розділити великі сторінки на менші, багаторазово використовувані компоненти, наприклад, окремі файли для навігаційного меню, футера чи рекламних банерів.

1. Фільтр include

Що робить include?

Фільтр include дозволяє підключити вміст одного шаблону в інший. Він допомагає підтримувати код чистим і організованим.

Синтаксис:

```
{% include "шлях_до_файлу" %}
```

Приклад:

Припустимо, у нас є простий HTML-файл banner.html:

```
<div>Посібник з Django на HELLO.COM</div>
```

Тепер ми можемо підключити цей банер у наш основний шаблон index.html:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Django на HELLO.COM</title>
</head>
<body>
    <h2>Головна сторінка</h2>
    {% include "banner.html" %}
</body>
</html>
```

При завантаженні сторінки index.html Django автоматично вставить вміст banner.html у вказане місце.

Використання змінних

Замість жорсткого шляху до файлу ви можете використовувати змінну, яка містить ім'я шаблону.

- **Приклад:**

```
{% include template_name %}
```

У цьому випадку template_name повинна бути змінною, переданою з представлення.

2. Передача даних у вкладені шаблони

Використання with

Ви можете передати конкретні дані у вкладений шаблон за допомогою оператора with.

- **Приклад у banner.html:**

```
<div>Посібник з {{ tutorial }} на {{ site }}</div>
```

- **Приклад у index.html:**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Django на HELLO.COM</title>
</head>
<body>
    <h2>Головна сторінка</h2>
    {% include "banner.html" with tutorial="Python" site="HELLO.COM" %}
</body>
</html>
```

Тут ми явно передаємо змінні tutorial та site у шаблон banner.html.

Автоматична передача даних

За замовчуванням, вкладені шаблони можуть використовувати змінні з батьківського шаблону.

- **Приклад у views.py:**

```
from django.shortcuts import render

def index(request):
    return render(request, "index.html", context={"site": "HELLO.COM"})
```

- **Приклад у index.html:**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Django на HELLO.COM</title>
</head>
<body>
    <h2>Головна сторінка</h2>
    <!-- Змінна `site` автоматично передається -->
    {% include "banner.html" with tutorial="Python" %}
</body>
</html>
```

Навіть якщо ми не вказали site у with, він буде доступний у banner.html, оскільки був переданий у index.html.

Обмеження передачі даних (only)

Якщо ви хочете заборонити автоматичну передачу даних із батьківського шаблону, використовуйте оператор only. Це допомагає уникнути конфліктів імен змінних.

- **Приклад:**

```
{% include "banner.html" with tutorial="Python" only %}
```

Комплексне завдання: Створення веб-сторінки "Профіль користувача"

Вам потрібно створити просту сторінку "Профіль користувача", яка демонструє використання тегів, фільтрів, статичних файлів та вкладених шаблонів.

Частина 1: Налаштування та статичні файли

1. **Налаштуйте структуру:**
   - У вашому застосунку Django створіть теку `templates/` з підтекою, названою так само, як ваш застосунок (наприклад, `templates/myapp/`).
   - Створіть у застосунку теку `static/` з підтеками `css/` та `images/`.
2. **Створіть статичні файли:**
   - У `static/css/` створіть файл `profile.css`.
   - У `static/images/` додайте будь-яке зображення для аватара користувача.
3. **Налаштуйте `settings.py`:**
   - Переконайтеся, що `django.contrib.staticfiles` є у вашому `INSTALLED_APPS`.
   - Перевірте, що `STATIC_URL` дорівнює `'static/'`.

Частина 2: Дані та `views.py`

1. **Створіть дані користувача:**
   - У файлі `views.py` створіть функцію-представлення `profile(request)`.
   - Створіть словник `user_data`, який міститиме інформацію про користувача:
     - `name`: рядок з ім'ям
     - `is_active`: булеве значення `True` або `False`
     - `bio`: довгий рядок з текстом, що містить якісь HTML-теги (наприклад, `<b>`, `<i>`).
     - `last_login`: об'єкт `datetime` з поточною датою і часом.
     - `interests`: список рядків.
2. **Передайте дані в шаблон:**
   - У функції `profile` використайте `render` для передачі `user_data` до шаблону `profile.html`.

Частина 3: Шаблони та теги

1. **Створіть шаблони:**
   - Створіть базовий шаблон `base.html` з блоками `title` та `content`. Додайте загальний футер з поточною датою.

    - Створіть шаблон `profile.html`, який розширює `base.html` (`{% extends "base.html" %}`).
    - Створіть окремий файл `user_info.html`, який буде містити лише інформацію про ім'я та статус.
2. **Використайте теги та фільтри:**
    - У `profile.html` використайте `{% load static %}`.
    - У `base.html` у футері використайте тег `{% now %}` для відображення поточної дати.
    - У `profile.html` використайте `{% if %}` для відображення статусу `is_active` (наприклад, "Активний" або "Неактивний").
    - Використайте `{% for %}` для ітерації та виведення списку інтересів.
    - Використайте фільтр `|cut` для видалення певного слова з біографії.
    - Використайте фільтр `|floatformat` для округлення числа (можна додати його в словник `user_data` як `rating`).
    - Використайте фільтр `|date` для форматування дати останнього входу.
    - Використайте фільтр `|capfirst` для відображення першої літери імені з великої літери.
3. **Включення шаблонів (`include`):**
    - У `profile.html` використайте тег `{% include "user_info.html" %}` для вставки інформації про користувача.
    - Спробуйте передати ім'я користувача до `user_info.html` двома способами:
        - Автоматично (з батьківського шаблону).
        - Використовуючи `with` та `only`.

Частина 4: Маршрутизація

1. **Налаштуйте `urls.py`:**
    - Створіть маршрут для вашої сторінки профілю.
    - Додайте маршрути для сторінок `about` та `contacts`, використовуючи `TemplateView` з `extra_context`.

Це завдання охоплює всі основні теми, і його виконання допоможе вам закріпити знання на практиці. Успіхів!