Django Starter

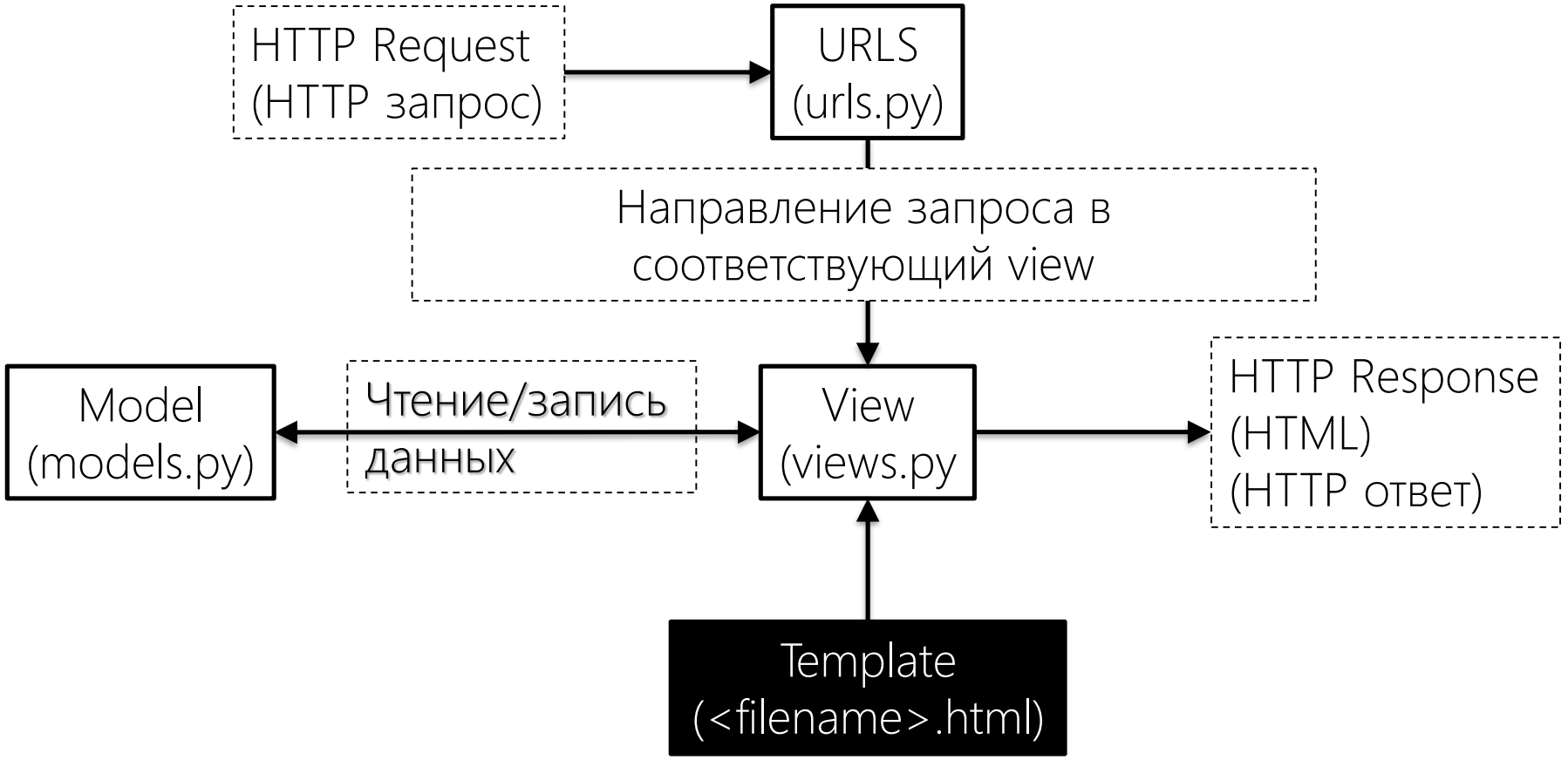
Шаблоны

Шаблоны

План урока

1. Шаблонизатор Django
2. Способы загрузки шаблонов
3. Синтаксис шаблонов: переменные, теги, фильтры и комментарии
4. Шаблонизатор Jinja: что это и зачем он может быть нужен

Структура файлов в реализации MVC в Django (MTV)



Шаблонизатор Django

Шаблонизатор **Django** используется для динамической генерации HTML. Шаблоны содержат статический **HTML** и динамические данные в специальном синтаксисе.

Django предоставляет бэкенд для собственной системы шаблонов, которая называется - язык шаблонов **Django** (**Django template language**, **DTL**)

Также поддерживается шаблонизатор **Jinja2**.

Поддержка шаблонов и встроенная система шаблонов **Django** находятся в одном пакете django.template.

Настройка шаблонов в Django

Шаблоны можно настроить с помощью списка словарей TEMPLATES в settings.py, который содержит настройки для систем шаблонов. После startproject мы получаем:

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            # ... some options here ...
        },
    },
]
```

Пояснение настроек шаблонов в Django

BACKEND - Python путь к классу бэкенда шаблонов для импорта. Встроенные бэкенды это **django.template.backends.django.DjangoTemplates** – стандартный шаблонизатор и **django.template.backends.jinja2.Jinja2** – шаблонизатор **Jinja**.

DIRS - список каталогов с шаблонами. Поиск по порядку.

APP_DIRS – искать ли шаблоны в установленных приложениях.

OPTIONS содержит настройки специфические для бэкенда.

Возможна одновременная поддержка нескольких бэкендов.

Загрузка шаблонов в Django

Используются 2 функции для загрузки шаблонов: **get_template** и **select_template**. Обе находятся в библиотеке **django.template.loader.**

get_template загружает шаблон с указанным названием, **select_template** позволяет загружать из списка шаблонов. При этом загружается первый доступный шаблон из списка.

Есть 2 типа ошибок при загрузке:

- шаблон не найден – будет вызвано исключение TemplateDoesNotExist.
- шаблон найден, но синтаксис не верен – будет вызвано TemplateSyntaxError.

Синтаксис функций для загрузки шаблонов в Django

Синтаксис вызова:

```
get_template(template_name[, dirs][, using=None])
select_template(template_name_list[, dirs][, using=None])
```

В dirs можно переназначить DIRS из settings.py, если в этом есть необходимость.

Также можно указывать в template_name путь к вложенной папке через слеш, например:

```
get_template('news/story_detail.html')
```

Объект Template

get_template и **select_template**, в результате успешной загрузки шаблона, возвращают объект Template, у которого есть метод render со следующим синтаксисом:

Template.render(context=None, request=None)

Данный метод отображает найденный шаблон с указанным context.

context – это контекст типа dict, передаваемый в шаблон для отображения. Если это None, то шаблон будет отображен с пустым контекстом.

request – это объект класса django.http.HttpRequest, доступ к которому будет предоставлен в шаблоне.

render_to_string – автоматизация загрузки шаблонов

Т.к. процесс загрузки и отображения шаблонов обычно повторяется, то в Джанго есть метод **render_to_string**, который загружает шаблон как get_template или select_template, и сразу вызывает его метод render.
Синтаксис метода:

render_to_string(template_name, context=None, request=None, using=None)

Если template_name – str, то вызывается get_template, если это – list, то вызывается select_template. При этом context и request такие же, как у метода Template.render. Атрибут using позволяет задать имя бэкенда, который будет использоваться для обработки шаблонов.

Синтаксис шаблонов Джанго

Шаблон Django – это просто текстовый файл, или строка **Python**, которые следуют языку шаблонов **Django**. Определенные конструкции распознаются и интерпретируются шаблонизатором. Основные – это переменные и теги.

Шаблон рендерится с контекстом. Рендеринг заменяет переменные на их значения, которые ищутся в контексте, и выполняет теги. Все остальное выводится как есть.

Синтаксис языка шаблонов **Django** использует четыре конструкции:

- Переменные
- Теги
- Фильтры
- Комментарии

Переменные в шаблонах

Переменные выводят значения из контекста, который является словарем. В шаблоне переменные выделяются {{ и }}, например:
My first name is {{ first_name }}. My last name is {{ last_name }}.

Если мы зададим **context={'first_name': 'John', 'last_name': 'Doe'},** то шаблон отрендерит:
My first name is John. My last name is Doe.

Обращение к ключам словаря, атрибутам объектов и элементам списка выполняется через точку:
{{ my_dict.key }}, {{ my_object.attribute }}, {{ my_list.0 }}

Если значением переменной является вызываемый объект, шаблонизатор вызовет его без аргументов и подставит результат.

Теги в шаблонах

Теги позволяют добавлять произвольную логику в шаблон. Например, теги могут выводить текст, добавлять логические операторы, такие как "if" или "for", получать содержимое из базы данных, или предоставлять доступ к другим тегам.

Теги выделяются {% и %}, например:
{% csrf_token %}

Большинство тегов принимают аргументы:
{% cycle 'odd' 'even’ %}

Некоторые теги требуют закрывающий тег:
{% if user.is_authenticated %}Hello, {{ user.username }}.{% endif %}

Основные теги в Джанго 1

{% if user.is_authenticated %}Hello, {{ user.username }}.{% endif %} – отображение с условием.

Основные особенности: нельзя использовать скобочки и вложенные сравнения a > b > c.

{% if test_list %}test_list не пустой

{% elif test_bool %}test_list пустой, но test_bool – true

{% else %}test_list пустой и test_bool – false

{% endif %}

{% for obj in list %} ... {% endfor %} – несколько раз выполняет содержимое, подставляя вместо obj элементы list. Как обычный цикл.

{% for obj in list %} ... {% empty %} ... {% endfor %} – вариант цикла с отображением в случае пустого list: тогда отображается информация, заключенная после {% empty %}.

Основные теги в Джанго 2

{% cycle 'odd' 'even' %} – возвращает свои аргументы при каждом вызове. При первом будет вызван первый, при втором – второй и т.д., после последнего опять будет первый.

{% csrf_token %} – тег, используемый для csrf-защиты. Подробнее об этом в 9 уроке.

{% comment %} ... {% endcomment %} – тег, используемый для комментирования. Любой текст, который находится между этими тегами будет проигнорирован. Можно также задать строку-описание в первом теге:

{% comment "Комментарий с описанием" %} Тут может быть временно отключенный код {% endcomment %}

{% debug %} – тег, выводящий большое количество информации для отладки, включая текущий контекст шаблона и все импортированные модули.

Основные теги использования шаблонов в Джанго

{% include "foo/bar.html" %} – загрузить шаблон и отобразить его с контекстом текущего шаблона.

В **Джанго** поддерживается система наследования с помощью пары тегов **block** и **extends**:

{% block block_name %}Текст по умолчанию.{% endblock %} – задает блок, который может быть легко переопределен расширяющими (наследующими) шаблонами. Таким образом можно составить "скелет" сайта, а потом наполнять его контентом в зависимости от текущей страницы.

block_name – идентификатор, определяющий блок.

{% extends "base.html" %} – загружает указанный шаблон как родительский. При задании **{% block block_name %}Текст для отображения.{% endblock %}** – переопределит родительский block указанным текстом. Если какой-то блок не переопределен, то будет использован текст из родительского блока.

Количество уровней вложенности шаблонов друг в друга не ограниченно.

Фильтры в шаблонах Джанго

Фильтры преобразуют переменные и аргументы тегов. Указываются с помощью

Пример фильтров:
{{ django|title }}

Для контекста {'django': 'the web framework for perfectionists with deadlines'} этот шаблон выведет:
The Web Framework For Perfectionists With Deadlines
Т.е. этот фильтр вызовет метод title() у 'django' элемента и вернет результат.

Некоторые фильтры принимают аргументы:
{{ my_date|date:"Y-m-d" }}

Примеры использования фильтров 1

default – если значение False (None, "", 0), то подставляется значение из **default**:
{{ value|default:"nothing" }} если value это "" (пустая строка), то вывод будет **nothing.**

dictsort – принимает список словарей, и возвращает этот список отсортированным по ключу:
{{ value:dictsort: "name" }}

Если value:

```
[
    {'name': 'zed', 'age': 19},
    {'name': 'amy', 'age': 22},
    {'name': 'joe', 'age': 31},
]
```

То вывод будет такой:

```
[
    {'name': 'amy', 'age': 22},
    {'name': 'joe', 'age': 31},
    {'name': 'zed', 'age': 19},
]
```

Особенности использования фильтра dictsort 1

dictsort можно, также как и остальные фильтры, использовать в тегах, а также указывать значения вложенных ключей, например:

```
{% for book in books|dictsort:"author.age" %}
    * {{ book.title }} ({{ book.author.name }})
{% endfor %}
```

Если books:

```
[
    {'title': '1984', 'author': {'name': 'George', 'age': 45}},
    {'title': 'Timequake', 'author': {'name': 'Kurt', 'age': 75}},
    {'title': 'Alice', 'author': {'name': 'Lewis', 'age': 33}},
]
```

То вывод будет отсортирован по ключу [author][age]:

* Alice (Lewis)
* 1984 (George)
* Timequake (Kurt)

Особенности использования фильтра dictsort 2

dictsort также может сортировать список списков, если в качестве аргумента ему передать индекс (типа int):

```
{{ value|dictsort:0 }}
```

Если value:

```
[
    ('a', '42'),
    ('c', 'string'),
    ('b', 'foo'),
]
```

То вывод будет отсортирован по элементу 0:

```
[
    ('a', '42'),
    ('b', 'foo'),
    ('c', 'string'),
]
```

При этом индекс нельзя передавать строкой, т.е. так неправильно:

```
{{ value|dictsort:"0" }}
```

Фильтр будет пытаться обратиться к ключу словаря "0", и, в результате, выведет пустую строку.

Примеры использования фильтров 2

divisibleby – возвращает True, если значение делимо на аргумент без остатка.
{{ value|divisibleby:"3" }}

Если value равно 21, то вернется True.

Пример использования внутри тега:
{% if value|divisibleby: "4" %} Делится на 4 {% else %} Не делится на 4 {% endif %}

Floatformat – форматирует Float, округляя его до указанного количества символов. При этом если количество указать с минусом, то в отсутствии дробной части не будет возвращать 0. Аргумент по умолчанию -1, т.е. округляет до 1 символа после запятой, не отображая 0, если дробная часть 0:

value	Template	Output	value	Template	Output
34.23234	{{ value|floatformat }}	34.2	34.53234	{{ value|floatformat:0 }}	35
34.00000	{{ value|floatformat }}	34	34.00000	{{ value|floatformat:-3 }}	34
34.00000	{{ value|floatformat:3 }}	34.000	34.23234	{{ value|floatformat:-3 }}	34.232

Примеры использования фильтров 3

random – возвращает случайный элемент из списка, например если value равно ['a', 'b', 'c', 'd'], то может вернуться 'b'.

stringformat – форматирует значение в соответствии с аргументом, аналогично printf-style форматированию с помощью %, но без самого символа %:

```
{{ value|stringformat:"E" }}
```

Если значение value у нас 10, то вывод будет 1.000000E+01. Это эквивалентно записи "%e" % value.

linenumbers – отображает текст с номерами строк, например:

value	{{ value|linenumbers }}
one	1. one
two	2. two
three	3. three

Пример шаблона для представления

Составим шаблон, который будем использовать для представления.

У нас есть список:

latest_question_list = [{'id': 1, 'question_text': 'В чем смысл жизни?"}, {'id': 2, 'question_text': 'Что первично, дух или материя?"}, {'id': 3, 'question_text': 'Существует ли свобода воли?"}]

Нужно составить шаблон, который проверяет значение переменной latest_question_list, и, если этот список не пустой, то выводит его элементы question в виде HTML списка <li>, где указывается ссылка на "/polls/question.id/", с текстом **question.question_text**. Если список пустой, то показывается строка "Список вопросов пуст".

Пример шаблона для представления

```
{% if latest_question_list %}
    <ul>
    {% for question in latest_question_list %}
        <li><a href="/polls/{{ question.id }}/">{{ question.question_text }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>Список вопросов пуст.</p>
{% endif %}
```

Пример использования шаблона в представлении

```
from django.http import HttpResponse

from django.template import loader

from .models import Question


def index(request):
    latest_question_list = Question.objects.order_by('-pub_date')[:5]
    template = loader.get_template('polls/index.html')
    context = {'latest_question_list': latest_question_list }
    return HttpResponse(template.render(context, request))
```

Шаблонизатор Jinja: что это и зачем может быть нужен

- Синтаксис **Jinja2** сильно похож на **Django**-шаблонизатор, но при этом дает возможность использовать чистые **Python** выражения и поддерживает гибкую систему расширений.
- Также **Jinja2** поддерживается и другими фреймворками, а не только **Django**.
- Помимо прочего можно настроить **Django** так, чтобы в одном случае он использовал шаблонизатор **Django**, а для другого – **Jinja2**.

Всё это выходит за рамки этого курса, но, тем не менее, имейте в виду.

- Основное преимущество **Jinja** шаблонизатора – скорость, больше логики в шаблоны, универсальность (легче перенести на другие фреймворки).

Шаблонизатор Jinja: преимущества

- Основное преимущество – скорость.
- Кастомные шаблонные теги. Есть context processor, которые сам передаёт request и другие переменные в шаблон. Также в Jinja2 нет некоторых тегов, например cycle, но они легко заменяются другими командами/тегами.
- Jinja2 позволяет добавлять больше логики в шаблоны. Но принято считать, что шаблоны должны отвечать только за то, каким образом информация показывается пользователю. Тем не менее в реальном мире редко удается это реализовать полностью, поэтому иногда Jinja2 оказывается лучшим выбором, чем встроенная в Джанго template-система. Но вы всегда можете расширить ваши шаблоны или перевести часть из них на Jinja2.

План урока

1. Шаблонизатор Django
2. Способы загрузки шаблонов
3. Синтаксис шаблонов: переменные, теги, фильтры и коментарии.
4. Шаблонизатор Jinja: что это и зачем он может быть нужен

Проверка знаний

TestProvider.com

TestProvider – это online сервис проверки знаний по информационным технологиям. С его помощью Вы можете оценить Ваш уровень и выявить слабые места. Он будет полезен как в процессе изучения технологии, так и для общей оценки знаний IT специалиста.

Проверьте как Вы усвоили данный материал на TestProvider.com

Успешное прохождение финального тестирования позволит Вам получить соответствующий Сертификат.

Спасибо за внимание! До новых встреч!

Лазорык Михаил
Software developer

tieto

django

Информационный видеосервис для разработчиков программного обеспечения

ITVDN

IT VIDEO DEVELOPERS NETWORK

TestProvider PROMETRIC TEST CENTER CyberBionic systematics Microsoft Partner Silver Learning Windows Azure Cloud Partner EBA European Business Association