**Глава 2 Немного подробнее о шаблонах**

**Шаблоны:**

Шаблон представляет собой текстовый файл. Он может генерировать любой текстовый формат (HTML, XML, CSV и т. д.). Шаблон содержит переменные , которые заменяются значениями при оценке шаблона, и теги , управляющие логикой шаблона. Настройка функциональности шаблонов в проекте Django производится в файле settings.py. с помощью переменной TEMPLATES. Так, по умолчанию переменная TEMPLATES в файле settings.py имеет следующее определение:

***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',***

***],***

***},***

***},***

***]***

Данная переменная принимает список конфигураций для каждого движка шаблонов. По умолчанию определена одна конфигурация, которая имеет следующшие параметры:

**BACKEND**: движок шаблонов. По умолчанию применяется встроенный движок django.template.backends.django.DjangoTemplates

**DIRS**: определяет список каталогов, где движок шаблонов будет искать файлы шаблонов. По умолчанию пустой список

**APP\_DIRS**: указывает, будет ли движок шаблонов искать шаблоны внутри папок приложений в папке templates.

**OPTIONS**: определяет дополнительный список параметров

**Передача данных в шаблоны:**

Одним из преимуществ шаблонов является то, что мы можем передать в них динамически из представлений различные данные. Для вывода данных в шаблоне могут использоваться различные способы. Для вывода самых простых данных применяется двойная пара фигурных скобок:

**{{ название\_объекта }}**

Чтобы из функции-представления передать данные в шаблон применяется третий параметр функции render, который еще называется context и который представляет словарь. Например, изменим файл views.py следующим образом:

***from django.shortcuts import render***

***def index(request):***

***data = {"header": "Hello Django", "message": "Welcome to Python"}***

***return render(request, "index.html", context=data)***

В шаблоне используются две переменных, соответственно словарь, который передается в функцию render через параметр context, теперь содержит два значения с ключами header и message.

**Фильтры:**

Вы можете изменить переменные для отображения с помощью фильтров .

Фильтры выглядят так. Это отображает значение переменной после фильтрации через фильтр, который преобразует текст в нижний регистр. Используйте вертикальную черту ( |), чтобы применить фильтр**.{{ name | lower }}** или так: **{{ name }}[lower](https://docs.djangoproject.com/en/4.1/ref/templates/builtins/" \l "std-templatefilter-lower)|**

Фильтры могут быть «связаны». Выход одного фильтра применяется к следующему. — распространенная идиома для экранирования текстового содержимого с последующим преобразованием разрывов строк в теги**. {{ text|escape|linebreaks }}<p>**

Некоторые фильтры принимают аргументы. Аргумент фильтра выглядит так: . Это отобразит первые 30 слов переменной**. {{ bio|truncatewords:30 }}bio**

Аргументы фильтра, содержащие пробелы, должны быть заключены в кавычки; например, чтобы присоединиться к списку с запятыми и пробелами, которые вы бы использовали **.{{ list|join:", " }}** . С полным списком фильтров можно ознакомиться по ссылке:[**https://docs.djangoproject.com/en/4.1/ref/templates/builtins/#ref-templates-builtins-filters**](https://docs.djangoproject.com/en/4.1/ref/templates/builtins/#ref-templates-builtins-filters)

**Теги:**

Теги более сложны, чем переменные: некоторые создают текст в выводе, некоторые управляют потоком, выполняя циклы или логические операции, а некоторые загружают внешнюю информацию в шаблон для использования последующими переменными**. {% tag %}** Для некоторых тегов требуются начальный и конечный теги (например, ). **{% tag %}** ... tag contents ... **{% endtag %}**

**тэг for:**

Перебрать каждый элемент массива. Например, чтобы отобразить список спортсменов, представленный в athlete\_list:

***<ul>***

***{% for athlete in athlete\_list %}***

***<li>{{ athlete.name }}</li>***

***{% endfor %}***

***</ul>***

**тэг if:**

Вычисляет переменную, и если эта переменная «истинна», отображается содержимое блока:

***{% if athlete\_list %}***

***Number of athletes: {{ athlete\_list|length }}***

***{% elif athlete\_in\_locker\_room\_list %}***

***Athletes should be out of the locker room soon!***

***{% else %}***

***No athletes.***

***{% endif %}***

**Наследование шаблонов фильтр extends**

Нередко шаблоны должны иметь одинаковую базовую структуру, одни и те же блоки, при этом определять для отдельных блоков различное содержимое. Это позволяет сформировать единообразный стиль сайта, когда веб-страницы имеют одни и те же структурные элементы - меню, хедер, футер, сайдбары и так далее.

В этом случае мы можем определять все шаблоны по отдельности. Однако если возникнет необходимость изменить какой-то блок, например, добавить в общее меню еще один пункт, тогда придется менять все шаблоны, коих может быть довольно много. И в этом случае оптимальнее повторно использовать один базовый шаблон, который определяет все основные блоки.

Давайте рассмотрим наследование шаблонов, начав с примера:

***<html lang="en">***

***<head>***

***<link rel="stylesheet" href="style.css">***

***<title>{% block title %}My amazing site{% endblock %}</title>***

***</head>***

***<body>***

***<div id="sidebar">***

***{% block sidebar %}***

***<ul>***

***<li><a href="/">Home</a></li>***

***<li><a href="/blog/">Blog</a></li>***

***</ul>***

***{% endblock %}***

***</div>***

***<div id="content">***

***{% block content %}{% endblock %}***

***</div>***

***</body>***

***</html>***

Этот шаблон, который мы назовем **base.html,** определяет скелет HTML-документа, который можно использовать для страницы с двумя столбцами. Задача «дочерних» шаблонов — заполнить пустые блоки содержимым.

С помощью элементов {% block название\_блока %}{% endblock название\_блока %} определяются отдельные блоки шаблонов. При этом для каждого блока определяется открывающий элемент {% block название\_блока %} и закрывающий элемент {% endblock название\_блока %}.

Например, блок title:

***{% block title %}Default title{% endblock title %}***

Когда другие шаблоны будут применять данный шаблон, то они могут определить для блока title какое-то свое содержимое. Для каждого блока можно определить содержимое по умолчанию. Так, для блока title это строка "Default title". И если другие шаблоны, которые будут использовать данный шаблон, не определят содержимое для блока title, то данный блок будет использовать строку "Default title"

Дочерний шаблон может выглядеть так:

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

***{% block title %}My amazing blog{% endblock %}***

***{% block content %}***

***{% for entry in blog\_entries %}***

***<h2>{{ entry.title }}</h2>***

***<p>{{ entry.body }}</p>***

***{% endfor %}***

***{% endblock %}***

Тег [extends](https://docs.djangoproject.com/en/4.1/ref/templates/builtins/#std-templatetag-extends) является ключевым здесь. Он сообщает механизму шаблонов, что этот шаблон «расширяет» другой шаблон. Когда система шаблонов оценивает этот шаблон, сначала она находит родителя — в данном случае «base.html».

Таким образом, подобная функциональность позволяет нам избежать повторения одних и тех же элементов в разных шаблонах. И в то же время если нам потребуется что-то изменить, например, структуру страницы или какой-то общий элемент, то достаточно это сделать в одном файле - base.html.

**Вложенные шаблоны и фильтр include**

Фильтр include позволяет добавить в один шаблон другой шаблон. Название подключаемого шаблона передается фильтру в качестве параметра:

***{% include подключаемый\_файл %}***

Допустим, в приложении у нас имеются два шаблона: **index.html** и **banner.html**

Пусть в файле banner.html будет какой-нибудь простейший код:

***<div>Руководство по Django</div>***

В другом файле - index.html подключим данный шаблон:

***<!DOCTYPE html>***

***<html>***

***<head>***

***<meta charset="utf-8" />***

***<title>Django on METANIT.COM</title>***

***</head>***

***<body>***

***<h2>Главная страница</h2>***

***{% include "banner.html" %}***

***</body>***

***</html>***

В итоге при обращении к шаблону index.html на веб-странице появится содержимое из подключенного шаблона banner.html

Стоит отметить, что вместо жесткой установки пути к вложенному шаблону в виде строки, можно использовать переменные:

***{% include template\_name %}***

В данном случае предполагается, что template\_name - это переменная, которая хранит путь к вложенному шаблону.

**Статические файлы:**

Веб-приложение, как правило, использует различные статические файлы - изображения, файлы стилей css, скриптов javascript и так далее. Рассмотрим, как мы можем использовать подобые файлы.

При создании проекта Django он уже имеет некоторую базовую настройку для работы со статическими файлами. В частности, в файле settings.py определена переменная STATIC\_URL, которая хранит путь к каталогу со статическими файлами:

***STATIC\_URL = 'static/'***

Переменная STATIC\_URL имеет значение "static/", а это значит, что нам достаточно создать в папке приложения каталог с именем "static" и добавить в него необходимые нам статические файлы. Но, естественно, при необходимости через данную настройку мы можем изменить расположение каталога статических файлов***.***

Для загрузки статических файлов на html страницу необходимо следующее: в начале файла шаблона необходимо определить инструкцию

***{% load static %}***

При этом данный код должен идти после тега DOCTYPE.

Для определения пути к статическим файлам используются выражения типа:

***{% static "путь к файлу внутри папки static" %}***

***<img src="{% static 'my\_app/example.jpg' %}" alt="My image">***

**Настройка путей к файлам:**

Если нас не устраивает хранение файлов в каталоге по умолчанию - каталоге static, либо мы хотим указать несколько папок, то мы можем в файле settings.py задать все необходимые каталоги с помощью переменной STATICFILES\_DIRS, которая принимает список путей:

***STATICFILES\_DIRS = [***

***BASE\_DIR / "static",***

***"/var/www/static/",***

***"/somefolder/"***

***]***

**Задание:**

Использую готовые шаблоны bootstrap5 **https://getbootstrap.com/docs/5.0/examples/** создать каркас сайта для бронирования отелей ! На сайте должны быть представлены следующие страницы. Для выполнения задания использовать два словаря ***hotels = [{“name”:”some name”,”address”:”some address”,”stars”:4}]***, ***users : [{“name”:”some name”,”age”:22, comment = [“comment 1”,”comment 2”]}]***

1. Главная страница – с общей информацией о работе сайта и возможностях.
2. Страница заведениями и их подробной информацией( название, местоположение, количество звёзд)
3. Страница с пользователями и информацией о их имени и возрасте
4. Страница с комментариями пользователей.
5. \*Сделать возможным переход со страницы на страницу используя navbar а также организовать расширение базового шаблона
6. \*Для каждого пользователя скачать фотку и при выводе информации о пользователе выводить ещё и фото.