Мега-Учебник Flask, Часть 2: Шаблоны перевод tutorial

[Python](https://habrahabr.ru/hub/python/)\*, [Разработка веб-сайтов](https://habrahabr.ru/hub/webdev/)\*

Это вторая статья в серии, где я описываю свой опыт написания веб-приложения на Python с использованием микрофреймворка Flask.  
  
Цель данного руководства — разработать довольно функциональное приложение-микроблог, которое я за полным отсутствием оригинальности решил назвать microblog.

Оглавление

Краткое повторение

Если вы следовали инструкциям в первой части, то у вас должно быть полностью работающее, но еще очень простое приложение с такой файловой структурой:

microblog\

flask\

<файлы виртуального окружения>

app\

static\

templates\

\_\_init\_\_.py

views.py

tmp\

run.py

Для запуска приложения вы запускаете скрипт run.py, затем открываете url [http://localhost:5000](http://localhost:5000/) в вашем браузере.  
  
Мы продолжим с того момента, где мы остановились, так что вам следует убедиться, что вышеупомянутое приложение правильно установлено и работает.

Зачем нам нужны шаблоны

Рассмотрим то, как мы можем расширить наше маленькое приложение.  
Мы хотим, чтобы на главной странице нашего приложения микроблогов был заголовок, который приветствует вошедшего в систему пользователя, что весьма стандартно для приложений такого рода. Пока что игнорируем тот факт, что у нас нет пользователей в приложении, я предоставлю решение этой проблемы в нужный момент.  
  
Простым средством вывода большого и красивого заголовка было бы сменить нашу функцию представлений на выдачу html, примерно так:

**from** app **import** app

**@app.route('/')**

**@app.route('/index')**

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

user = { 'nickname': 'Miguel' } *# выдуманный пользователь*

**return** '''

<html>

<head>

<title>Home Page</title>

</head>

<body>

<h1>Hello, ''' + user['nickname'] + '''</h1>

</body>

</html>

'''

Попробуйте посмотреть как выглядит приложение в вашем браузере.  
  
Пока у нас нет поддержки пользователей, тем не менее, я обратился к использованию прототипов модели пользователя, которые иногда называют фальшивыми или мнимыми прототипами. Это позволяет нам сконцентрироваться на некоторых аспектах нашего приложения, зависящих от частей системы, которые еще не были написаны.  
  
Надеюсь вы согласитесь со мной, что решение выше очень уродливое. Представьте насколько сложным станет код, если вы должны возвращать громоздкую HTML страницу с большим количеством динамического содержимого. И что если вам нужно сменить макет вашего веб-сайта в большом приложении с множеством представлений, которые возвращают HTML напрямую? Это очевидно не масштабируемое решение.

Шаблоны спешат на помощь

Не задумывались ли вы о том, что если бы вы могли держать раздельно логику вашего приложения и макет, или представление ваших страниц было бы организовано куда лучше? Вы даже можете нанять веб-дизайнера, чтобы создать сногсшибательный сайт в то время, пока вы программируете его [сайта] поведение при помощи Python. Шаблоны помогут осуществить это разделение.  
  
Напишем наш первый шаблон (файл app/templates/index.html):

<html>

<head>

<title>{{title}} - microblog</title>

</head>

<body>

<h1>Hello, {{user.nickname}}!</h1>

</body>

</html>

Как видно выше, мы просто написали стандартную HTML страничку, но с одним лишь отличием: плейсхолдеры для динамического содержимого заключены в секции {{… }}  
  
Теперь рассмотрим использование шаблона в нашей функции представления (файл app/views.py):

**from** flask **import** render\_template

**from** app **import** app

**@app.route('/')**

**@app.route('/index')**

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

user = { 'nickname': 'Miguel' } *# выдуманный пользователь*

**return** render\_template("index.html",

title = 'Home',

user = user)

Запустите приложение на данном этапе, чтобы посмотреть как работают шаблоны. Если в вашем браузере отрисована страница, то вы можете сравнить ее исходный код с оригинальным шаблоном.  
  
Чтобы отдать страницу, нам нужно импортировать из Flask новую функцию под названием render\_template. Эта функция принимает имя шаблона и список переменных аргументов шаблона, а возвращает готовый шаблон с замененными аргументами.  
  
Под капотом: функция render\_template вызывает шаблонизатор Jinja2, который является частью фреймворка Flask. Jinja2 заменяет блоки {{...}} на соответствующие им значения, переданные как аргументы шаблона.

Управляющие операторы в шаблонах

Шаблоны Jinja2 помимо прочего поддерживают управляющие операторы, переданные внутри блоков {%...%}. Давайте добавим оператор if в наш шаблон (файл app/templates/index.html):

<html>

<head>

{% if title %}

<title>{{title}} - microblog</title>

{% else %}

<title>Welcome to microblog</title>

{% endif %}

</head>

<body>

<h1>Hello, {{user.nickname}}!</h1>

</body>

</html>

Теперь наш шаблон слегка поумнел. Если мы забудем определить название страницы в функции представления, то взамен исключения шаблон предоставит нам собственное название. Вы спокойно можете удалить аргумент заголовка из вызова render\_template в нашей функции представления, чтобы увидеть как работает оператор if.

Циклы в шаблонах

Вошедший в наше приложение пользователь наверняка захочет увидеть недавние записи от пользователей из его контакт-листа на главной странице, давайте посмотрим как же это сделать.  
  
В начале, мы провернем трюк, чтобы создать несколько ненастоящих пользователей и несколько записей для отображения (файл app/views.py):

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

user = { 'nickname': 'Miguel' } *# выдуманный пользователь*

posts = [ *# список выдуманных постов*

{

'author': { 'nickname': 'John' },

'body': 'Beautiful day in Portland!'

},

{

'author': { 'nickname': 'Susan' },

'body': 'The Avengers movie was so cool!'

}

]

**return** render\_template("index.html",

title = 'Home',

user = user,

posts = posts)

Чтобы отобразить пользовательские записи мы используем список, где у каждого элемента будут поля автор и основная часть. Когда мы доберемся до осуществления реальной базы данных мы сохраним эти имена полей, так что мы можем разрабатывать и тестировать наш шаблон, используя ненастоящие объекты, не беспокоясь об их обновлении, когда мы перейдем на базу данных.  
  
Со стороны шаблона мы должны решить новую проблему. Имеющийся список может содержать любое количество элементов и нужно решить сколько сообщений будет представлено. Шаблон не может сделать никаких предположений о количестве сообщений, поэтому он должен быть готов отобразить столько сообщений, сколько пошлет представление.  
  
Посмотрим как сделать это, используя управляющую структуру (файл app/templates/index.html):

<html>

<head>

{% if title %}

<title>{{title}} - microblog</title>

{% else %}

<title>microblog</title>

{% endif %}

</head>

<body>

<h1>Hi, {{user.nickname}}!</h1>

{% for post in posts %}

<p>{{post.author.nickname}} says: <b>{{post.body}}</b></p>

{% endfor %}

</body>

</html>

Не так уж и трудно, правда? Проверим приложение и обязательно проверим добавление нового содержимого в список записей.

Наследование шаблонов

Мы охватим еще одну тему, прежде чем закончить на сегодня.  
  
Нашему приложению микроблогов необходима навигационная панель с несколькими ссылками сверху страницы. Там будут ссылки на редактирование вашего профиля, выход и т.д.  
  
Мы можем добавить навигационную панель в наш шаблон index.html, но, как только наше приложение разрастется, нам понадобится больше шаблонов, и навигационную панель нужно будет скопировать в каждый из них. Тогда вы должны держать все эти копии одинаковыми. Это может стать трудоемкой задачей, если у вас много шаблонов.  
  
Вместо этого мы можем использовать наследование в шаблонах Jinja2, которое позволяет нам переместить части макета страницы в общий для всех базовый шаблон, из которого все остальные шаблоны будут наследоваться.  
  
Теперь определим базовый шаблон, который включает в себя навигационную панель, а также логику заголовка, которую мы реализовали ранее (файл app/templates/base.html):

<html>

<head>

{% if title %}

<title>{{title}} - microblog</title>

{% else %}

<title>microblog</title>

{% endif %}

</head>

<body>

<div>Microblog: <a href="/index">Home</a></div>

<hr>

{% block content %}{% endblock %}

</body>

</html>

В этом шаблоне мы использовали управляющий оператор block для определения места, куда могут быть вставлены дочерние шаблоны. Блокам даются уникальные имена.   
  
Теперь осталось изменить наш index.html так, чтобы он наследовался от base.html (файл app/templates/index.html):

{% extends "base.html" %}

{% block content %}

<h1>Hi, {{user.nickname}}!</h1>

{% for post in posts %}

<div><p>{{post.author.nickname}} says: <b>{{post.body}}</b></p></div>

{% endfor %}

{% endblock %}

Теперь только шаблон base.html отвечает за общую структуру страницы. Мы убрали те элементы отсюда и оставили только часть с содержимым. Блок extends устанавливает наследственную связь между двумя шаблонами, таким образом Jinja2 знает: если нам нужно отдать index.html, то нужно включить его в base.html. Два шаблона имеют совпадающие операторы block с именем content, именно поэтому Jinja2 знает как cкомбинировать два шаблона в один. Когда мы будем писать новые шаблоны, то также станем создавать их как расширения base.html

Заключительные слова

Если вы хотите сэкономить время, то приложение микроблога в текущей стадии доступно здесь:  
Скачать [microblog-0.2.zip](https://github.com/miguelgrinberg/microblog/archive/v0.2.zip)  
  
Учтите, что zip файл не содержит в себе виртуального окружения flask. Создайте его сами, следуя инструкциям в первой части, после этого вы сможете запустить приложение.  
  
В следующей части серии мы взглянем на формы. Надеюсь увидимся.  
  
Мигель