Desenvolvimento WEB com Python e Flask

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Câmpus Rio Grande
Divisão de Computação

Agenda

Debug

Para habilitar a publicação de erros no navegador em aplicações Flask:

```
app.debug = True
app.run()
```

Parâmetros GET

```python
@app.route('/user/<username>')
def show_user_profile(username):
    # show the user profile for that user
    return 'User %s' % username

@app.route('/post/<int:post_id>')
def show_post(post_id):
    # show the post with the given id, the id is an integer
    return 'Post %d' % post_id
```

Há os seguintes conversores:

int	aceita inteiros
float	como *int* para números de ponto flutuante
path	como padrão mas aceita barras

Parâmetros POST

Para acessar os valores vindos por POST, você deve acessar o atributo *form* do objeto *request*:

```
from flask import request
@app.route('/login', methods=['POST', 'GET'])
def login():
    error = None
    if request.method == 'POST':
        if valid_login(request.form['username'],
                       request.form['password']):
            return log_the_user_in(request.form['username'])
        else:
            error = 'Invalid username/password'
    # the code below is executed if the request method
    # was GET or the credentials were invalid
    return render_template('login.html', error=error)
```

Não esqueça de importar *request*

Redirecionamento e Erros

```python
from flask import abort, redirect, url_for

@app.route('/')
def index():
    return redirect(url_for('login'))

@app.route('/login')
def login():
    abort(401)
    this_is_never_executed()
```

```python
from flask import render_template

@app.errorhandler(404)
def page_not_found(error):
    return render_template('page_not_found.html'), 404
```

Trabalhando com Sessões

```
from flask import Flask, session, redirect, url_for, escape, request
app = Flask(__name__)

@app.route('/')
def index():
    if 'username' in session:
        return 'Logged in as %s' % escape(session['username'])
    return 'You are not logged in'

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        session['username'] = request.form['username']
        return redirect(url_for('index'))
    return '''
        <form action="" method="post">
            <p><input type=text name=username>
            <p><input type=submit value=Login>
        </form>
    '''
@app.route('/logout')
def logout():
    # remove the username from the session if it's there
    session.pop('username', None)
    return redirect(url_for('index'))
# set the secret key.  keep this really secret:
app.secret_key = 'A0Zr98j/3yX R~XHH!jmN]LWX/,?RT'
```

Trabalhando com Sessões

O método *pop()* irá deletar da sessão o valor com a determinada chave passada por parâmetro

```
@app.route('/logout')
def logout():
    session.pop('logged_in', None)
    flash('You were logged out')
    return redirect(url_for('show_entries'))
```

Sessão SecretKey

A *secretKey* deve possuir o valor mais aleatório possível.

Seu S.O tem bibliotecas que permitem a geração de um boa chave aleatória:

```
>>> import os
>>> os.urandom(24)
'\xfd{H\xe5<\x95\xf9\xe3\x96.5\xd1\x010<!\xd5\xa2\xa0\x9fR"\xa1\xa8'
```

É apenas copiar e colar a chave e está feito.

Templates

```python
from flask import render_template

@app.route('/hello/')
@app.route('/hello/<name>')
def hello(name=None):
    return render_template('hello.html', name=name)
```

Templates

Os arquivos Templates podem ficar localizados nas seguintes pastas:

Case 1: a module:

```
/application.py
/templates
    /hello.html
```

Case 2: a package:

```
/application
    /__init__.py
    /templates
        /hello.html
```

Templates

As marcações definidas são acessadas da seguinte forma:

```
<!doctype html>
<title>Hello from Flask</title>
{% if name %}
  <h1>Hello {{ name }}!</h1>
```

Templates

É possível acoplar dentro um template outros templates:

templates.html

```
<!doctype html>
<title>Flaskr</title>
<link rel=stylesheet type=text/css href="{{ url_for('static', filename='st
<div class=page>
  <h1>Flaskr</h1>
  <div class=metanav>
  {% if not session.logged_in %}
    <a href="{{ url_for('login') }}">log in</a>
  {% else %}
    <a href="{{ url_for('logout') }}">log out</a>
  {% endif %}
  </div>
  {% for message in get_flashed_messages() %}
    <div class=flash>{{ message }}</div>
  {% endfor %}
  {% block body %}{% endblock %}
</div>
```

Templates

show_entries.html

```
{% extends "layout.html" %}
{% block body %}
  {% if session.logged_in %}
    <form action="{{ url_for('add_entry') }}" method=post class=add-entry>
      <dl>
        <dt>Title:
        <dd><input type=text size=30 name=title>
        <dt>Text:
        <dd><textarea name=text rows=5 cols=40></textarea>
        <dd><input type=submit value=Share>
      </dl>
    </form>
  {% endif %}
  <ul class=entries>
  {% for entry in entries %}
    <li><h2>{{ entry.title }}</h2>{{ entry.text|safe }}
  {% else %}
    <li><em>Unbelievable.  No entries here so far</em>
  {% endfor %}
  </ul>
{% endblock %}
```

Templates

login.html

```
{% extends "layout.html" %}
{% block body %}
  <h2>Login</h2>
  {% if error %}<p class=error><strong>Error:</strong> {{ error }}{% endif %}
  <form action="{{ url_for('login') }}" method=post>
    <dl>
      <dt>Username:
      <dd><input type=text name=username>
      <dt>Password:
      <dd><input type=password name=password>
      <dd><input type=submit value=Login>
    </dl>
  </form>
{% endblock %}
```

Localização dos Arquivos Estáticos

Aplicações dinâmicas também precisam de arquivos estáticos:

- css
- js...

```
url_for('static', filename='style.css')
```

É preciso criar uma pasta *static*.

Message Flashing

```
from flask import Flask, flash, redirect, render_template, \
     request, url_for

app = Flask(__name__)
app.secret_key = 'some_secret'

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    error = None
    if request.method == 'POST':
        if request.form['username'] != 'admin' or \
                request.form['password'] != 'secret':
            error = 'Invalid credentials'
        else:
            flash('You were successfully logged in')
            return redirect(url_for('index'))
    return render_template('login.html', error=error)

if __name__ == "__main__":
    app.run()
```

Message Flashing

```
<!doctype html>
<title>My Application</title>
{% with messages = get_flashed_messages() %}
  {% if messages %}
    <ul class=flashes>
    {% for message in messages %}
      <li>{{ message }}</li>
    {% endfor %}
    </ul>
  {% endif %}
{% endwith %}
{% block body %}{% endblock %}
```

Message Flashing

```
{% extends "layout.html" %}
{% block body %}
  <h1>Overview</h1>
  <p>Do you want to <a href="{{ url_for('login') }}">log in?</a>
{% endblock %}
```

Message Flashing

```
{% extends "layout.html" %}
{% block body %}
  <h1>Login</h1>
  {% if error %}
    <p class=error><strong>Error:</strong> {{ error }}
  {% endif %}
  <form action="" method=post>
    <dl>
      <dt>Username:
      <dd><input type=text name=username value="{{
```

Trabalhando com Ajax

```
<script type=text/javascript>
  $(function() {
    $('a#calculate').bind('click', function() {
      $.getJSON($SCRIPT_ROOT + '/_add_numbers', {
        a: $('input[name="a"]').val(),
        b: $('input[name="b"]').val()
      }, function(data) {
        $("#result").text(data.result);
      });
      return false;
    });
  });
</script>
<h1>jQuery Example</h1>
<p><input type=text size=5 name=a> +
   <input type=text size=5 name=b> =
   <span id=result>?</span>
<p><a href=# id=calculate>calculate server side</a>
```

Trabalhando com Ajax

```
<script type=text/javascript>
  $SCRIPT_ROOT = {{ request.script_root|tojson|safe }};
</script>
```

Trabalhando com Ajax

```
from flask import Flask, jsonify, render_template, request
app = Flask(__name__)

@app.route('/_add_numbers')
def add_numbers():
    a = request.args.get('a', 0, type=int)
    b = request.args.get('b', 0, type=int)
    return jsonify(result=a + b)

@app.route('/')
def index():
    return render_template('index.html')
```

Upload de Arquivos

É possível listar os tipos de arquivos permitidos no upload e a pasta que ficará os arquivos do upload

```
import os
from flask import Flask, request, redirect, url_for
from werkzeug import secure_filename

UPLOAD_FOLDER = '/path/to/the/uploads'
ALLOWED_EXTENSIONS = set(['txt', 'pdf', 'png', 'jpg', 'jpeg', 'gif'])

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
```

Upload de Arquivos

Exemplo:

```
from flask import request
from werkzeug import secure_filename

@app.route('/upload', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        f = request.files['the_file']
        f.save('/var/www/uploads/' + secure_filename(f.filename))
    ...
```

Exemplo

```
def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1] in ALLOWED_EXTENSIONS

@app.route('/', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        file = request.files['file']
        if file and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
            return redirect(url_for('uploaded_file',
                                    filename=filename))
    return '''
    <!doctype html>
    <title>Upload new File</title>
    <h1>Upload new File</h1>
    <form action="" method=post enctype=multipart/form-data>
      <p><input type=file name=file>
         <input type=submit value=Upload>
    </form>
    '''
```

Outro Exemplo

```
from flask import request

@app.route('/upload', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        f = request.files['the_file']
        f.save('/var/www/uploads/uploaded_file.txt')
    ...
```

Outro Exemplo

Se você quer saber o nome que foi dado ao arquivo que o cliente enviou você pode aceessar o atributo **filename**. Todavia, tenha em mente que o valor pode ser forjado. Se você querer usar o nome do arquivo salvo em seu servidor, passe a usar o método **secure_filename**

```
from flask import request
from werkzeug import secure_filename

@app.route('/upload', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        f = request.files['the_file']
        f.save('/var/www/uploads/' + secure_filename(f.filename))
```

Upload de Arquivos

```
from flask import send_from_directory

@app.route('/uploads/<filename>')
def uploaded_file(filename):
    return send_from_directory(app.config['UPLOAD_FOLDER'],
                               filename)
```

Como forma alternativa, você pode registrar o *upload_file* como regra *build_only* e utilizar o **SharedDataMiddleware** (isso funciona também em versões antigas do Flask)

```
from werkzeug import SharedDataMiddleware
app.add_url_rule('/uploads/<filename>', 'uploaded_file',
                 build_only=True)
app.wsgi_app = SharedDataMiddleware(app.wsgi_app, {
    '/uploads':  app.config['UPLOAD_FOLDER']
})
```

Upload de Arquivos

```
from flask import request

@app.route('/upload', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        f = request.files['the_file']
        f.save('/var/www/uploads/uploaded_file.txt')
    ...
```

Definindo Configurações de Upload

É possível definir o tamanho máximo dos arquivos enviados por uploads:

```
from flask import Flask, Request

app = Flask(__name__)
app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024
```

Acesso Externo

Se você rodar o servidor notará que o mesmo está acessível somente em seu computador.

- Esse é o comportamento *default* de aplicações escritas em Python executadas em seu computador
- Todavia, se você tem o *debug* desativado e confia nos usuários de sua rede, você pode fazer com que outros usuários acessem sua aplicação.
  - Para que isso ocorra realize a seguinte alteração na chamada do método *run*:

```
app.run(host='0.0.0.0')
```

Deploy

Sua aplicação Flask está pronta? Se sim, você pode imediatamente hospedá-la nas seguintes plataformas:

- Deploying Flask on Heroku
- Deploying WSGI on dotCloud com Flask-specific notes
- Deploying Flask on Webfaction
- Deploying Flask on Google App Engine
- Sharing your localhost Server com Localtunnel

Callbacks - After e Before

```python
@app.after_request
def call_after_request_callbacks(response):
    for callback in getattr(g, 'after_request_callbacks', ()):
        callback(response)
    return response
```

```python
from flask import request

@app.before_request
def detect_user_language():
    language = request.cookies.get('user_lang')
    if language is None:
        language = guess_language_from_request()
        @after_this_request
        def remember_language(response):
            response.set_cookie('user_lang', language)
    g.language = language
```

Callbacks - After e Before

After customizado

```
from flask import g
def after_this_request(func):
    if not hasattr(g, 'call_after_request'):
        g.call_after_request = []
    g.call_after_request.append(func)
    return func

@app.after_request
def per_request_callbacks(response):
    for func in getattr(g, 'call_after_request', ()):
        response = func(response)
    return response
```

```
def invalidate_username_cache():
    @after_this_request
    def delete_username_cookie(response):
        response.delete_cookie('username')
        return response
```

Desenvolvimento WEB com Python e Flask

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Câmpus Rio Grande
Divisão de Computação