Webservices mit Python Flask

Alexander Böhm
alexander.boehm@malbolge.net

Chemnitz Linux-Tage, 11. März 2018

Motivation

- Einfache GUI
- Geht das auch im Browser?
- Ich brauch mal eine Webapplikation
- Verschiedene Datenquellen zusammenführen
- Geht das auch schnell? einfach? unkompliziert?

Flask ist eine Lösung.

- Webframework für Python 2/3
- Schneller Einsteig, übersichtlich
- Webserver *werkzeug*
- Template Engine *jinja2*
- Einen Zoo von Erweiterungen
- Website flask.pocoo.org

Was brauche ich dafür?

- Eine Plattform, die Python unterstützt
- Flask selbst, installierbar über pip:

```
pip install flask
```

- Und los geht's

Hello World

hello_world.py:

```
1   # Flask importieren
2   from flask import Flask
3
4   # Flask-Applikation initialisieren
5   app = Flask(__name__)
6
7   # Route zum Index festlegen
8   @app.route('/')
9
10  # Behandlungsmethode definieren
11  def index():
12      return 'Hello, World!'
13
14  if __name__ == '__main__':
15      # Applikation starten
16      app.run()
```

Starten der Applikation

- Über Flask

```
FLASK_APP=hello_world.py flask run
```

- oder direkt

```
python hello_world.py
```

Hello World via JSON

hello_world_json.py:

```
1  from flask import Flask, jsonify
2  app = Flask(__name__)
3
4
5  @app.route('/')
6  def hello_world():
7      # dict in JSON umwandeln
8      return jsonify({'message': 'Hello, World!'})
9
10
11 if __name__ == '__main__':
12     app.run()
```

Anfragenmethode eingrenzen

```
1  from flask import Flask, request
2  app = Flask(__name__)
3
4
5  # Nur POST-Anfragen erlauben
6  @app.route('/', methods=['POST'])
7  def only_post_allowed():
8      # Parameter auslesen
9      param1 = request.args.get('param1')
10
11     return 'It\'s a POST with arg1 set to %s' % (param1)
```

Parameter

```
1   # Untypisierter Parameter
2   @app.route('/any/<param_any>')
3   def any_param(param_any):
4       return '%s is not casted' % (str(param))
5
6   # Zahl als Parameter
7   @app.route('/number/<int:param_number>')
8   def a_number(param_number):
9       return '%i is a number' % (param_number)
10
11  # String als Parameter
12  @app.route('/string/<string:param_string>')
13  def a_string(param_string):
14      return '%s is a string' % (param_string)
```

Fehlerbehandlung

error_handling.py:

```
1  from flask import Flask, abort, redirect, url_for
2  app = Flask(__name__)
3
4  # Behandlung des Statuscodes 404
5  @app.errorhandler(404)
6  def error_not_allowed(e):
7      return redirect(url_for('index'))
8
9  # Die Indexseite
10 @app.route('/')
11 def index():
12     return 'Hello world!'
13
14 # Zugriff auf diese Seite soll verboten sein
15 @app.route('/secret')
16 def not_allowed():
17     abort(401)
```

Templates (1/2)

template.py:

```
1  from flask import Flask, render_template
2  app = Flask(__name__)
3
4
5  @app.route('/')
6  def index():
7      my_list = ["Item 1", "Item 2", "Item 3"]
8      # Template unter 'templates/index.html' rendern
9      # 'my_list' wird als Variable bekannt gemacht
10     return render_template('index.html', my_list=my_list)
11
12
13 if __name__ == '__main__':
14     app.run()
```

Templates (2/2)

templates/index.html:

```
1  <html>
2  <head>
3    <title>Hello world!</title>
4  </head>
5  <body>
6    <!-- Laufe über Variable 'my_list' -->
7    {% for item in my_list %}
8      <!-- Gib das aktuelle Element in 'item' aus -->
9      {{ item }}<br/>
10   {% endfor %}
11 </body>
12 </html>
```

Blueprints (1/2)

- Problem: Größere Applikationen
  - → steigende Komplexität
  - → unübersichtlich

Blueprints (1/2)

- Problem: Größere Applikationen
  - → steigende Komplexität
  - → unübersichtlich
- Lösung: Blueprints
  - Modularisierung
  - Kapselung von Funktionen, Templates, etc.
  - Wiederverwendung von Funktionen

Blueprints (2/2)

```
1  from flask import Blueprint, render_template
2
3  # Lege Blueprint an
4  page_blueprint = Blueprint('page', __name__,
5                             template_folder='templates/page')
6
7  # Lege Routen für Blueprint fest
8  @page_blueprint.route('/show/<int:page>')
9  def page_show(page):
10     #  Nutze Template des Blueprints
11     return render_template('show.html', page=page)
12
13 ...
14 # Registriere Blueprint mit Prefix 'page'
15 app.register_blueprint(page_blueprint,
16                         url_prefix='/page')
```

Kontakt für Nachfragen

Alexander Böhm

alexander.boehm@malbolge.net

Fingerprint: C2AA 3A42 66D1 11B2 7C37 74EB 2438 B8AD FDF4 5447

Github @aboehm