Web service in Python with Flask

Tim Guicherd - ACIT2515

Flask

- Numerous libraries in Python for creating web platforms / APIs
- We will use `Flask` : http://flask.pocoo.org/
- Simple and easy to use
- Excellent documentation / tutorial for Flask: https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world
- Install it with `pip install flask`

What is Flask?

- `Flask` is a microframework: it does most of the heavy job of dealing with HTTP requests / responses.
- It has additional support for automatic JSON serialization / deserialization.
- It is very easy to use, and has a development server: you can easily run and test your code on your computer.
- All you need is:

```
from flask import Flask

app = Flask(__name__)

if __name__ == "__main__"
    app.run(debug=True)
```

> Use `debug=True` to set up automatic live reload and informative error pages.

Working with Flask: routes

- To work with your application, use the decorator `@app.route("/something")`.
- The return value of the function will be sent as the HTTP response.

Example:

```
@app.route("/example")
def example():
    return "I am running Flask!"
```

- The function above defines what happens when someone tries to browse to `/example`.

Templates

- You could render the whole HTML page in the `return` statement, but it is very impractical.
- A better way is to use the `render_template` method. It allows to create an HTML template. The template can contain variables and basic logic, and will be interpolated at runtime.
- Use `render_template('my_template.html', variable1=value1)`.
- Flask will look for `my_template.html` in the `templates` folder.
- In that template, all references to `{{ variable1 }}` will be replaced by `value1`.

Logic in templates

- It is possible to insert some logic in the templates, instead of the views. This allows to reuse the same template for different views.
- Flask uses `Jinja2` for its templates. The documentation has many useful examples.

A simple Flask app with a template

- `templates/base.html`:

```
<html>
  <body>
    <!-- The value will change depending on the variable sent to the template -->
    Hello, {{ username }}! Here are some numbers from 1 to 5:
    <ul>{% for x in range(1, 6) %}<li>{{ x }}</li>{% endfor %}</ul>
  </body>
</html>
```

- `app.py`

```
from flask import Flask, render_template

app = Flask(__name__)
@app.route('/')
def homepage():
    return render_template('base.html', username="John Doe")

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

Use dictionary unpacking to make your `render_template` calls easier to read

```
render_template("index.html", var1=value1, var2=value2)

# The following is better and easier to manage
context = {"var1": value1, "var2": value2}
render_template("index.html", **context)
```

Routes with arguments

- You may want to define a route that takes an argument.
- For instance `/student/1234/` that displays information about student 1234.

```
@app.route("/student/<int:student_id>")
def show_student(student_id):
    return render_template(...)
```

- Notice how the function now takes an argument: the value received in the URL.
- The list of converters is available on Flask's website

Flask converters for URL rules

Item	Type
string	Default type: accepts any text without slash
int	Positive integers
float	Positive integers
path	Like strings, but accepts slashes
uuid	Accepts UUID strings

Reversing URLs in templates or Python code

In Python

```
from flask import url_for

link = url_for("show_student", student_id=1234)
```

- `url_for` takes the **name of the function**, and all keyword arguments required

In a Jinja template

```
<a href="{{ url_for('show_student', student_id=1234) }}">Student 1234</a>
```

Template scaffolding

- As much as you can reuse code in Python, you can create reusable templates.
- Generally, there is a "base" template that other templates can build on.

Example (`templates/base.html`)

```
<html>
<head><title>Example</title></head>
<body><div class="container">
{% block main %}
{% endblock %}
</div></body>
</html>
```

Child template

- In `templates/home.html`

```
{% extends "base.html" %}
{% block main %}
<h1>Welcome to my Flask website</h1>
{% endblock %}
```

- In `templates/student.html`

```
{% extends "base.html" %}
{% block main %}
<h1>Student {{ student_id }}</h1>
{% endblock %}
```

HTTP error codes with Flask

- In a view, you can also return a tuple:
  - the first element is the response
  - the second element is the status code

```
@app.route('/data/<value>')
def show_data(value):
  if value == "notfound":
    # We specify the status code in the return value
    return render_template('error.html'), 404
  else:
    # By default, Flask uses 200 (or 204)
    return render_template('page.html')
```