Flask

Marcin Jenczmyk

Clearcode

m.jenczmyk@clearcode.cc

27/05/2017

Overview

requirements.txt

Requirements file is a plaintext file listing Python `pip` dependencies for a project.

requirements.txt

Requirements file is a plaintext file listing Python `pip` dependencies for a project.

To install dependencies from `requirements.txt` into current Python envrinoment run

```
doctor@TARDIS:~$ pip install -r requirements.txt
```

requirements.txt

Requirements file is a plaintext file listing Python `pip` dependencies for a project.

To install dependencies from `requirements.txt` into current Python envrinoment run

```
doctor@TARDIS:~$ pip install -r requirements.txt
```

To save list of Python packages installled in current Python envrinoment into `requirements.txt` file run

```
doctor@TARDIS:~$ pip freeze > requirements.txt
```

requirements.txt

```
appdirs==1.4.3
click==6.7
Flask==0.12.2
itsdangerous==0.24
Jinja2==2.9.6
MarkupSafe==1.0
packaging==16.8
pyparsing==2.2.0
six==1.10.0
Werkzeug==0.12.2
```

Figure: A sample requirements file.

Hello there!

Flask is a microframework for Python for a web development.

Hello there!

Flask is a microframework for Python for a web development.
Some useful resources:

- `http://flask.pocoo.org/`
- `http://flask.pocoo.org/docs/latest/quickstart/`
- `http://flask.pocoo.org/extensions/`
- `https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world`

Hello there!

Flask is a microframework for Python for a web development.
Some useful resources:

- http://flask.pocoo.org/
- http://flask.pocoo.org/docs/latest/quickstart/
- http://flask.pocoo.org/extensions/
- https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world

```
doctor@TARDIS:~$ pip install Flask
```

Hello there!

```python
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello there!"

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

Figure: A simple Flask app (see `hello_v1/hello.py`).

Hello there!

To run a Flask application run

```
doctor@TARDIS:~$ python hello.py
```

Hello there!

To run a Flask application run

```
doctor@TARDIS:~$ python hello.py
```

Debug mode

It's easier to debug application behaviour in debug mode - to do this add `app.debug = True` in your Python code or export `FLASK_DEBUG` envrinoment variable

```
doctor@TARDIS:~$ export FLASK_DEBUG=1
```

Hello there!

To run a Flask application run

```
doctor@TARDIS:~$ python hello.py
```

Debug mode

It's easier to debug application behaviour in debug mode - to do this add `app.debug = True` in your Python code or export `FLASK_DEBUG` envrinoment variable

```
doctor@TARDIS:~$ export FLASK_DEBUG=1
```

Warning

Debug mode should be never used on a production!

HTTP methods

To use a GET HTTP method put a `<type:arg>` in a view URL, where `type` can be either `string`, `int`, `float`, `path`, `any` (any of listed before) or `uuid`.

```
@app.route('/')
@app.route('/hello/<string:name>/')
def hello(name=None):
    if name is None:
        return 'Hello there!'
    elif name == 'there':
        return 'General Kenobi!'
    else:
        return 'Hello {}!'.format(name)
```

Figure: Sample view with GET parameter (see `hello_v2/hello.py`).

HTTP methods

To use POST HTTP method one has to enable it in a view decorator.

```
@app.route('/login/', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        # Do some stuff to log a user using POST data.
        pass
    else:
        # Render login form allowing to log in.
        pass
    return 'Login'
```

Figure: Sample view handling POST parameter (see `hello_v2/hello.py`).

HTTP methods

Remark

One can get URL to view by its name using `url_for` function, ex. one can get login view URL by calling `url_for('login')`.

Jinja2

Rendering entire HTML markup for a webpage by writing strings to be returned by views would be tiresome - there is a Jinja2 engine built in Flask to enable rendering HTML templates.

Jinja2

Rendering entire HTML markup for a webpage by writing strings to be returned by views would be tiresome - there is a Jinja2 engine built in Flask to enable rendering HTML templates.

One can render template using `render_template` function, **Flask will be looking for templates in the `templates` directory, located at the same path as Python application file!**

```
/
├── hello.py
└── templates
```

Jinja2

```
@app.route('/hello/<string:name>')
def hello(name=None):
    from flask import render_template

    if name == 'there':
        greetings = 'General Kenobi!'
    else:
        greetings = 'Hello {}!'.format(name or 'there'
                                       )

    return render_template(
        'index.html',
        greetings=greetings
    )
```

Figure: Sample view rendering Jinja template (see `hello_v3/hello.py`).

Jinja2

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Title of the document</title>
</head>

<body><h1>{{ greetings }}</h1></body>
</html>
```

Figure: Jinja template for hello_3 example (see `hello_v3/templates/index.html`).

Jinja2

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Title of the document</title>
  {% block css %}{% endblock %}
</head>

<body>{% block body %}{% endblock %}</body>
</html>
```

Figure: Jinja templates can be extended (see `index.html` and `base.html` - on next slide - in `hello_v4/templates/`).

Jinja2

```
{% extends "base.html" %}
{% block body %}
  <h1>{{ greetings }}</h1>
{% endblock %}
```

Jinja2

```
<ul id="navigation">
{% for item in navigation %}
  <li><a href="{{ item.href }}">
     {{ item.caption }}
  </a></li>
{% endfor %}
</ul>
```

```
{% if url %}
<a href="{{ url }}">Mysterious URL</a>
{% endif %}
```

Figure: Jinja templates allow for using `for` loops and `if` commands.

Static files

Dynamic web applications also need static files. They are going to be searched for in `static` directory (but on production envrinoment server should handle them); to generate URLs for static files, use the special `static` endpoint name.

Static files

Dynamic web applications also need static files. They are going to be searched for in `static` directory (but on production envrinoment server should handle them); to generate URLs for static files, use the special `static` endpoint name.

To generate URL for a static file use `url_for` function, ex. `url_for('static', filename='style.css')`.

```
/
├── hello.py
├── templates
└── static
```

Static files

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

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

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

Figure: Python code for `hello_v5` example app.

Static files

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

{% block css %}
  <link rel="stylesheet" type="text/css"
     href="{{␣url_for(
␣␣␣␣␣␣␣'static',␣filename='style.css')
␣␣}}">
{% endblock %}
```

Figure: Jinja template rendering static content (see `hello_v5/templates/index.html`, continuation on next slide).

Static files

```
{% block body %}
  <img class="meme"
    src="{{ url_for('static', filename='hello.jpg')
  }}">
{% endblock %}
```

Figure: Jinja template rendering static content, continuation (see `hello_v5/templates/index.html`).

References

Armin Ronacher (2017)
`http://flask.pocoo.org/`

Armin Ronacher (2008)
`http://jinja.pocoo.org/docs/2.9/`