



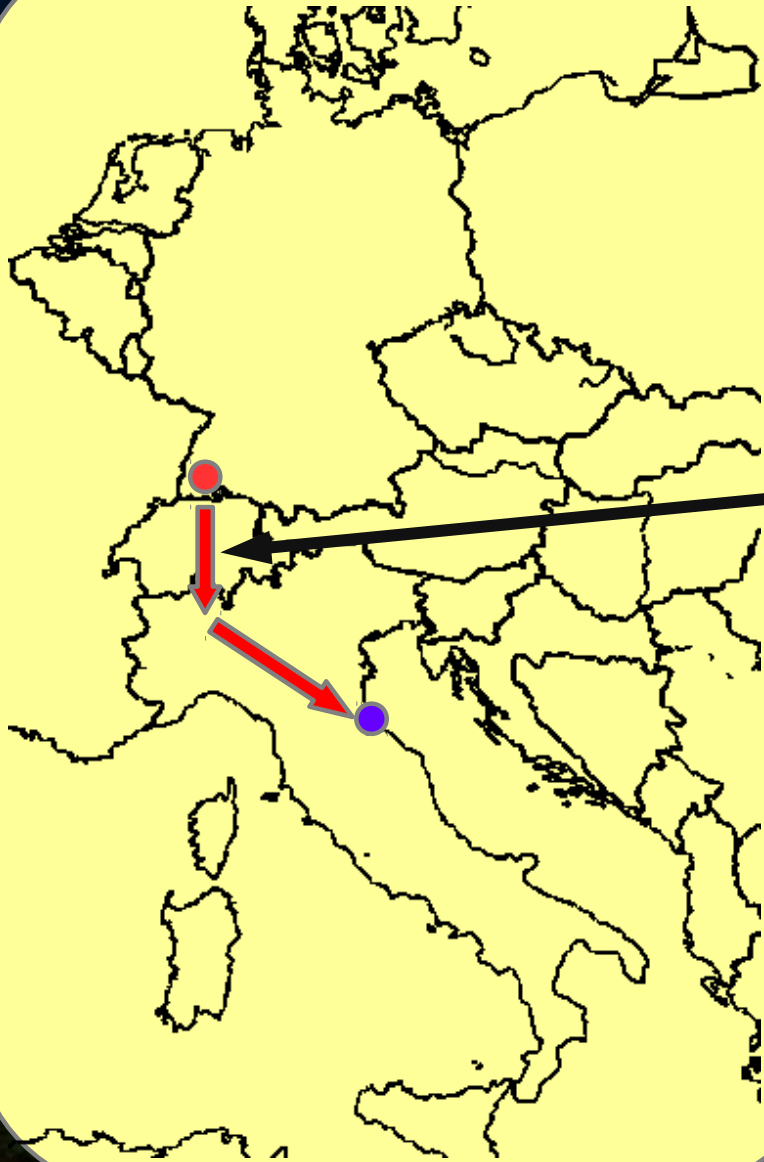
# When Django is too bloated

Specialized Web-Applications with Werkzeug

# Niklas Meinzer



@NiklasMM



Gotthard Base Tunnel



**mps**

Medizinische  
Planungssysteme

# Python is amazing for web developers!

The Django logo, featuring the word "django" in a white, lowercase, sans-serif font on a dark green rectangular background.

The web framework for  
perfectionists with deadlines.



# Flask

web development,  
one drop at a time

- Bottle
- BlueBream
- CherryPy
- CubicWeb
- Grok
- Nagare
- Pyjs
- Pylons
- TACTIC
- Tornado
- TurboGears
- web2py
- Webware
- Zope 2

# Why would I want to use less?

- Learn how stuff works





# Why would I want to use less?

- Avoid over-engineering
  - Wastes time and resources
  - Makes updates harder
  - It's a security risk.

The Telegraph

## Cyber attack hits German train stations as hackers target Deutsche Bahn



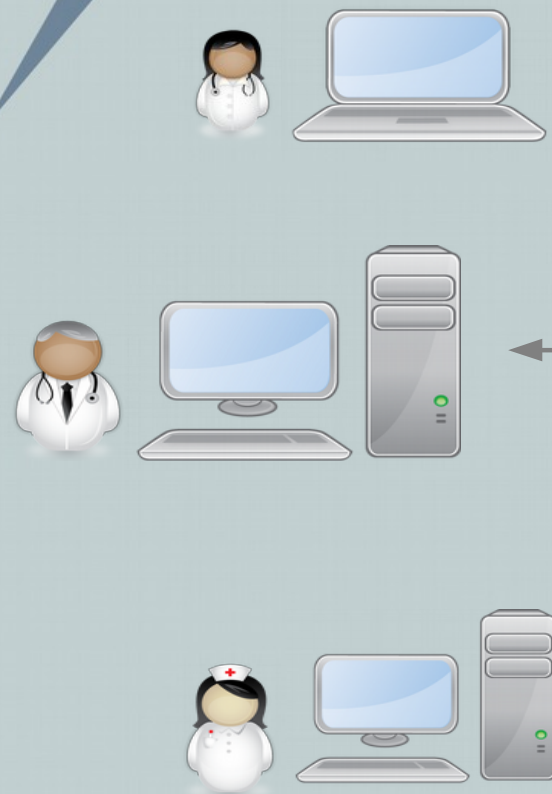
An information monitor at a German train station displays the ransomware message  
CREDIT: @ZECHENTATEN/TWITTER

# Why would I want to use less?

- You want to do something very specific



- Plan, manage and document chemotherapy treatments
- Built with modern web technology
- Used by hospitals in three European countries



REST



Database

Patient  
Data



HL7



Lab  
Data



Pharmacy System



Printers

# Werkzeug = German for “tool”



- Developed by pocoo team @ pocoo.org
  - Flask, Sphinx, Jinja2
- A “WSGI utility”
- Very lightweight
  - No ORM, No templating engine, etc
- The basis of Flask and others





# Werkzeug Features Overview

- WSGI
  - WSGI 1.0 compatible, WSGI Helpers
- Wrapping of requests and responses
- HTTP Utilities
  - Header processing, form data parsing, cookies
- Unicode support
- URL routing system
- Testing tools
  - Testclient, Environment builder
- Interactive Debugger in the Browser

## A simple Application

```
1 from werkzeug.wrappers import Request, Response
2
3 def application(environ, start_response):
4     request = Request(environ)
5     name = request.args.get('name', 'Rimini!')
6
7     response = Response("Buona sera, %s!" % name)
8     return response(environ, start_response)
9
10 if __name__ == '__main__':
11     from werkzeug.serving import run_simple
12     run_simple('localhost', 4000, application)
```

## A simple Application

```
1 from werkzeug.wrappers import Request, Response
2
3 @Request.application
4 def application(request):
5     name = request.args.get('name', 'Rimini!')
6     return Response("Buona sera, %s!" % name)
7
8 if __name__ == '__main__':
9     from werkzeug.serving import run_simple
10    run_simple('localhost', 4000, application)
```

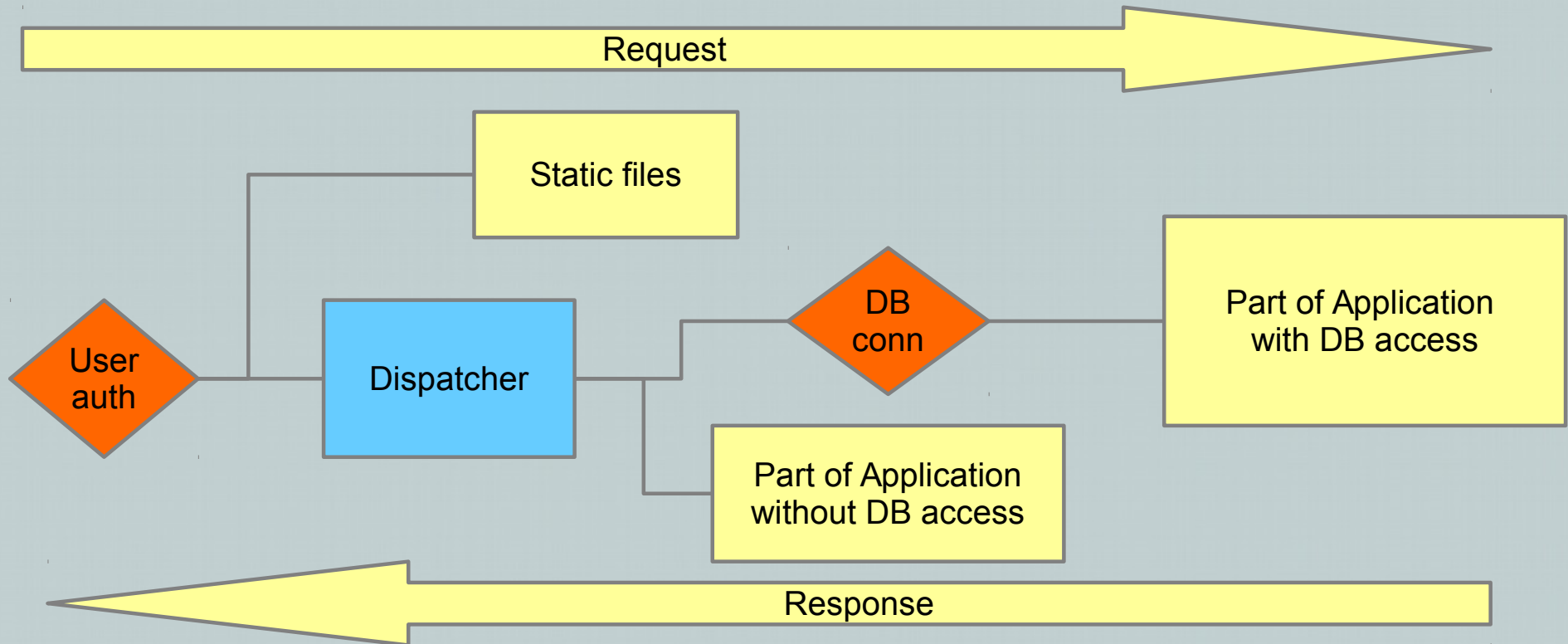


# URL Routing

```
1 from werkzeug.routing import Map, Rule
2
3 url_map = Map([
4     Rule('/', endpoint='EuroPython/overview'),
5     Rule('/<int:year>', endpoint='EuroPython/event'),
6     Rule(
7         '/<any(mon,tue,wed,thu,fri):day>', endpoint='EuroPython/schedule'
8     ),
9     Rule('/greeting/<name>', endpoint='greeter'),
10 ])
11
12 def application(environ, start_response):
13     urls = url_map.bind_to_environ(environ)
14     try:
15         endpoint, args = urls.match()
16     except HTTPException, e:
17         return e(environ, start_response)
18     start_response('200 OK', [('Content-Type', 'text/plain')])
19     return ['Rule points to %r with arguments %r' % (endpoint, args)]
```

# Middleware

- Separate parts of the Application as wsgi apps
- Combine as needed





```
1 from myproject.app import MyDatabaseApp, MyRegularApp
2 from myproject.auth import AuthenticationMiddleware
3 from myproject.db import DBConnectionMiddleware
4
5 from werkzeug.wsgi import DispatcherMiddleware, SharedDataMiddleware
6
7 from werkzeug.exceptions import NotFound
8
9 def build_application():
10
11     # build Database aware part
12     app = DBConnectionMiddleware(MyDatabaseApp)
13
14     # Use dispatcher to connect three parts of app
15     app = DispatcherMiddleware(
16         NotFound,
17         {
18             "/app": app,
19             "/noDBApp": MyRegularApp
20         }
21     )
22
23     # wrap static file server
24     app = SharedDataMiddleware(app, {
25         "static": "/var/www/staticfiles"
26     })
27
28     app = AuthenticationMiddleware(app)
29
30     return app
```

# HTTP Utilities

- Work with HTTP dates

```
Sun, 06 Nov 1994 08:49:37 GMT ; RFC 822, updated by RFC 1123
Sunday, 06-Nov-94 08:49:37 GMT ; RFC 850, obsoleted by RFC 1036
Sun Nov  6 08:49:37 1994      ; ANSI C's asctime() format
```

- Read and dump cookies
- Parse form data

```
>>> from cStringIO import StringIO
>>> data = '--foo\r\nContent-Disposition: form-data; name="test"\r\n' \
... '\r\nHello World!\r\n--foo--'
>>> environ = {'wsgi.input': StringIO(data), 'CONTENT_LENGTH': str(len(data)),
...           'CONTENT_TYPE': 'multipart/form-data; boundary=foo',
...           'REQUEST_METHOD': 'POST'}
>>> stream, form, files = parse_form_data(environ)
>>> stream.read()
''
>>> form['test']
u'Hello World!'
```

## Using the test client

```
1 from werkzeug.test import Client
2 from werkzeug.testapp import test_app
3 from werkzeug.wrappers import BaseResponse
4
5
6 def test_my_app():
7     c = Client(test_app, BaseResponse)
8     resp = c.get('/')
9     assert 200 == resp.status_code
10
11     assert "{ 'key': 'value' }" == resp.data
```



## Using the test client - pytest fixtures

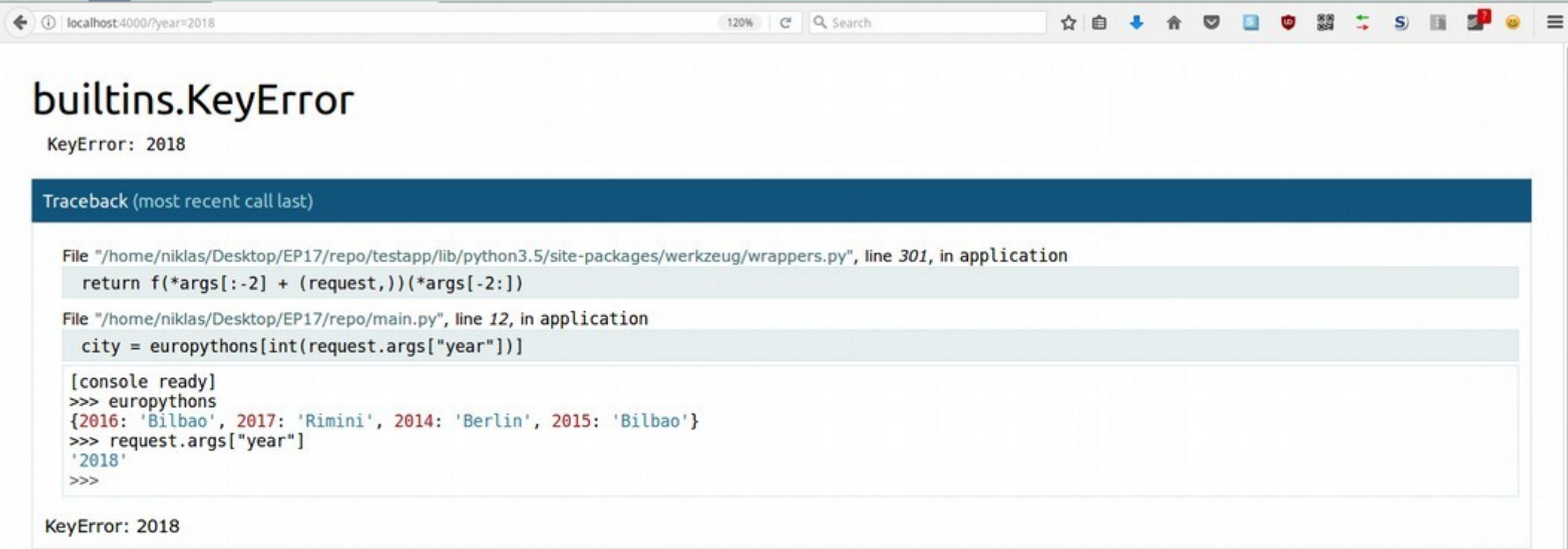
```
1 from werkzeug.test import Client
2 from werkzeug.testapp import test_app
3 from werkzeug.wrappers import BaseResponse
4
5 @pytest.fixture
6 def client():
7     return Client(test_app, BaseResponse)
8
9 def test_my_app(client):
10     resp = client.get('/')
11     assert 200 == resp.status_code
12
13     assert "{ 'key': 'value' }" == resp.data
```

# Using the test client - pytest fixtures

```
1 from werkzeug.test import Client
2 from werkzeug.testapp import test_app
3 from myproject.auth import AuthenticationMiddleware
4 from werkzeug.wrappers import BaseResponse
5
6 @pytest.fixture
7 def client():
8     return Client(test_app, BaseResponse)
9
10 @pytest.fixture
11 def auth_client():
12     app = AuthenticationMiddleware(test_app)
13     return Client(app, BaseResponse)
14
15 def test_my_app(auth_client):
16     resp = client.get('/')
17
18     # I'm not logged in =(
19     assert 401 == resp.status_code
```



# Interactive debugger in the Browser



The screenshot shows a web browser window with the address bar displaying 'localhost:4000/year=2018'. The page content shows a 'builtins.KeyError' exception with the message 'KeyError: 2018'. Below this, a 'Traceback (most recent call last)' section is visible, containing the following information:

- File `"/home/niklas/Desktop/EP17/repo/testapp/lib/python3.5/site-packages/werkzeug/wrappers.py", line 301, in application`  
`return f(*args[:-2] + (request,))(*args[-2:])`
- File `"/home/niklas/Desktop/EP17/repo/main.py", line 12, in application`  
`city = europythons[int(request.args["year"])]`

Below the traceback, a console window shows the following interaction:

```
[console ready]
>>> europythons
{2016: 'Bilbao', 2017: 'Rimini', 2014: 'Berlin', 2015: 'Bilbao'}
>>> request.args["year"]
'2018'
>>>
```

At the bottom of the traceback section, the message 'KeyError: 2018' is repeated.

The debugger caught an exception in your WSGI application. You can now look at the traceback which led to the error.

To switch between the interactive traceback and the plaintext one, you can click on the "Traceback" headline. From the text traceback you can also create a paste of it. For code execution mouse-over the frame you want to debug and click on the console icon on the right side.

You can execute arbitrary Python code in the stack frames and there are some extra helpers available for introspection:

- `dump()` shows all variables in the frame
- `dump(obj)` dumps all that's known about the object



# Endless possibilities

- Connect to a database with SQLAlchemy
  - Use Jinja2 to render documents
  - Use Celery to schedule asynchronous tasks
  - Talk to 3<sup>rd</sup> party APIs with requests
  - Make syscalls
- 
- Remote control a robot to perform tasks at home

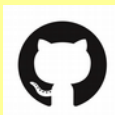




# Thank you!



@NiklasMM



NiklasMM

Photographer: Patrick Neumann