A Short History of Python Web Frameworks

Quazi Nafiul Islam

👋

I'm Nafiul!

@gamesbrainiac

@nafiul@fosstodon.org

sonar

sonarlint

sonarqube

Genesis

CHUCK

TALKING HEADS
ONCE IN A LIFETIME
17.743

And you may ask yourself, "How do I work this?"
And you may ask yourself, "Where is that large automobile?"
And you may tell yourself, "This is not my beautiful house"
And you may tell yourself, "This is not my beautiful wife"

And you may ask yourself, "How do I work this?"
And you may ask yourself, "Where is that large class?"
And you may tell yourself, "This is not my beautiful code"
And you may tell yourself, "This is not my beautiful server"
How did I get here?

I nerdsniped myself.

I started investigating something that had nothing to do with what I was supposed to be doing.

I learned a lot along the way.

I felt incredibly guilty.

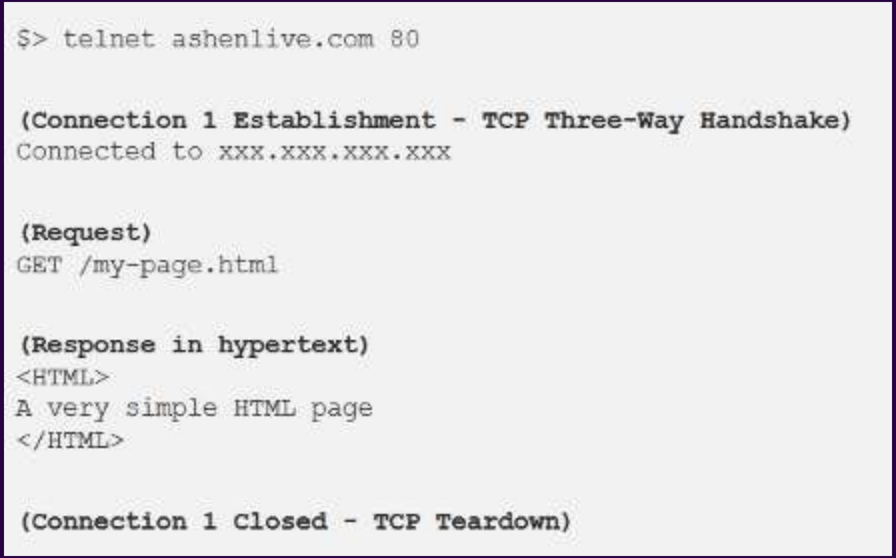
So, here’s a talk about what I learned.

Forgive me developers, for I have yak shaved.

1989

EINE WELT OHNE MILITAR ES GILT DIE MILITARISCH BEDINGTE
TEILUNG DURCH BURGERRECHTE DES WELTBURGERS ZU ERSETZEN
LONG LIVE DALAI LA
THIS WALL HAS

HTTP [1989]



```
$> telnet ashenlive.com 80

(Connection 1 Establishment - TCP Three-Way Handshake)
Connected to xxx.xxx.xxx.xxx

(Request)
GET /my-page.html

(Response in hypertext)
<HTML>
A very simple HTML page
</HTML>

(Connection 1 Closed - TCP Teardown)
```

1991: HTTP 0.9 (Look Ma! No Headers!)

Request

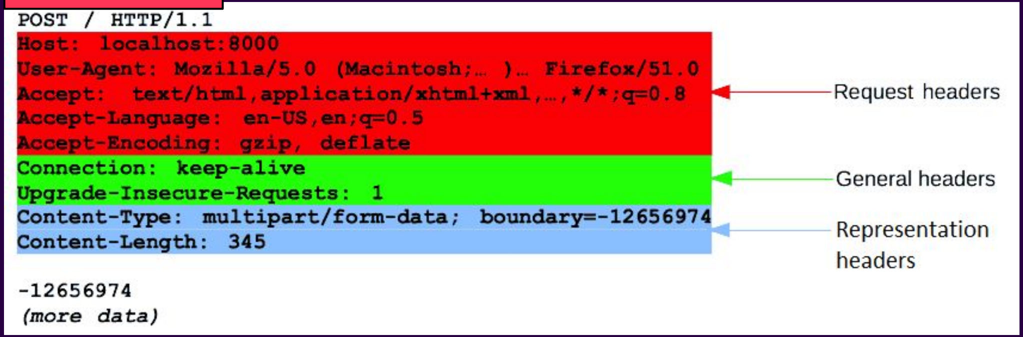
```
GET /4848 HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/3.01 (X11; I; SunOS 5.4 sun4m)
Pragma: no-cache
Host: tecfa.unige.ch:7778
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

Response

```
HTTP/1.0 200 OK
Date: Fri, 08 Aug 2003 08:12:31 GMT
Server: Apache/1.3.27 (Unix)
MIME-version: 1.0
Last-Modified: Fri, 01 Aug 2003 12:45:26 GMT
Content-Type: text/html
Content-Length: 2345
** a blank line *
<HTML> ...
```

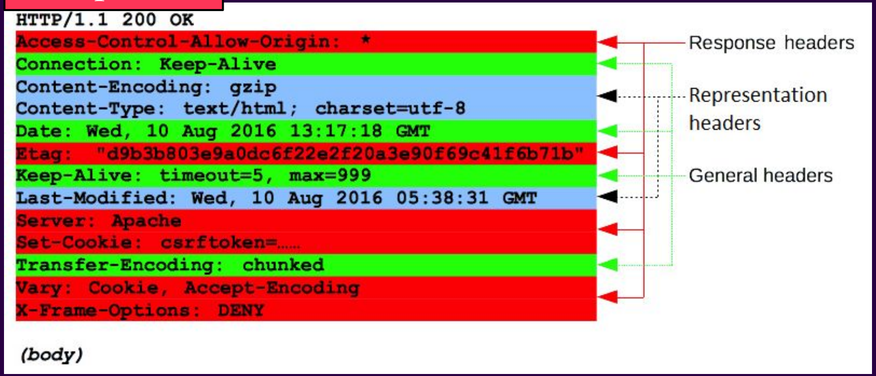
1991: HTTP 1.0 with Headers!

Request



```
POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh;… )… Firefox/51.0
Accept: text/html,application/xhtml+xml,…,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-12656974
Content-Length: 345

-12656974
(more data)
```

Response



```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Wed, 10 Aug 2016 13:17:18 GMT
Etag: "d9b3b803e9a0dc6f22e2f20a3e90f69c41f6b71b"
Keep-Alive: timeout=5, max=999
Last-Modified: Wed, 10 Aug 2016 05:38:31 GMT
Server: Apache
Set-Cookie: csrftoken=……
Transfer-Encoding: chunked
Vary: Cookie, Accept-Encoding
X-Frame-Options: DENY

(body)
```

1991: HTTP 1.1: What we mostly use today

HTTP Changes from 1.0 to 1.1

Hits

Byte Ranges: You could now request specific byte ranges.

Connection Persistence: Multiple HTTP requests could be sent over a single connection. (Keep-Alive)

Chunked Transfers: Can now start sending data without knowing the full size.

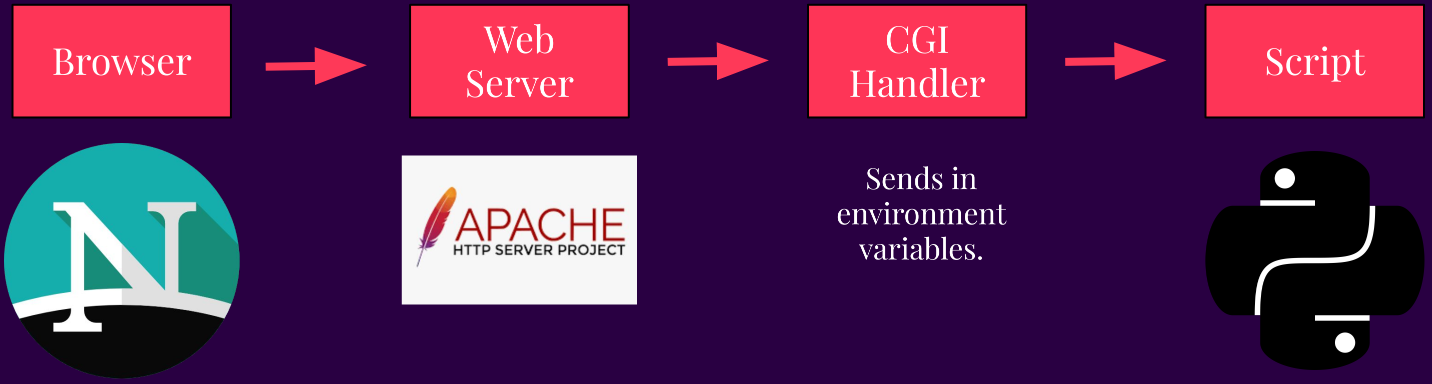
Misses

New HTTP Methods: New verbs introduced such as OPTIONS, PUT, DELETE and TRACE

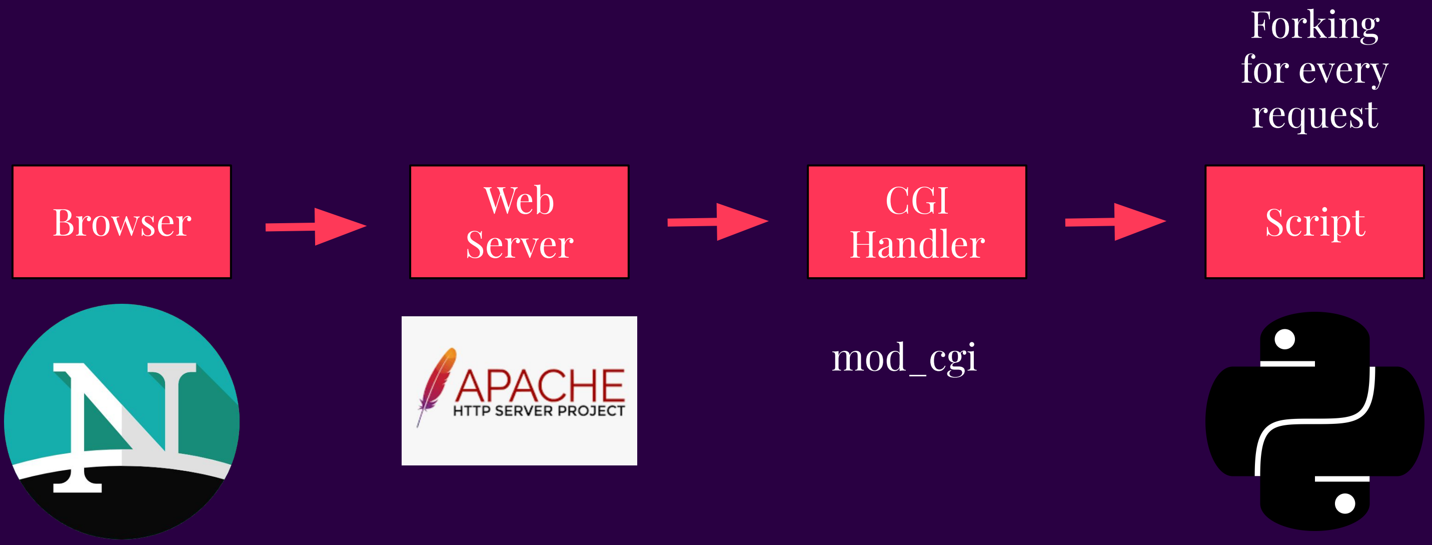
SOMETHING IS
MISSING
makeameme.org

We want
dynamic content

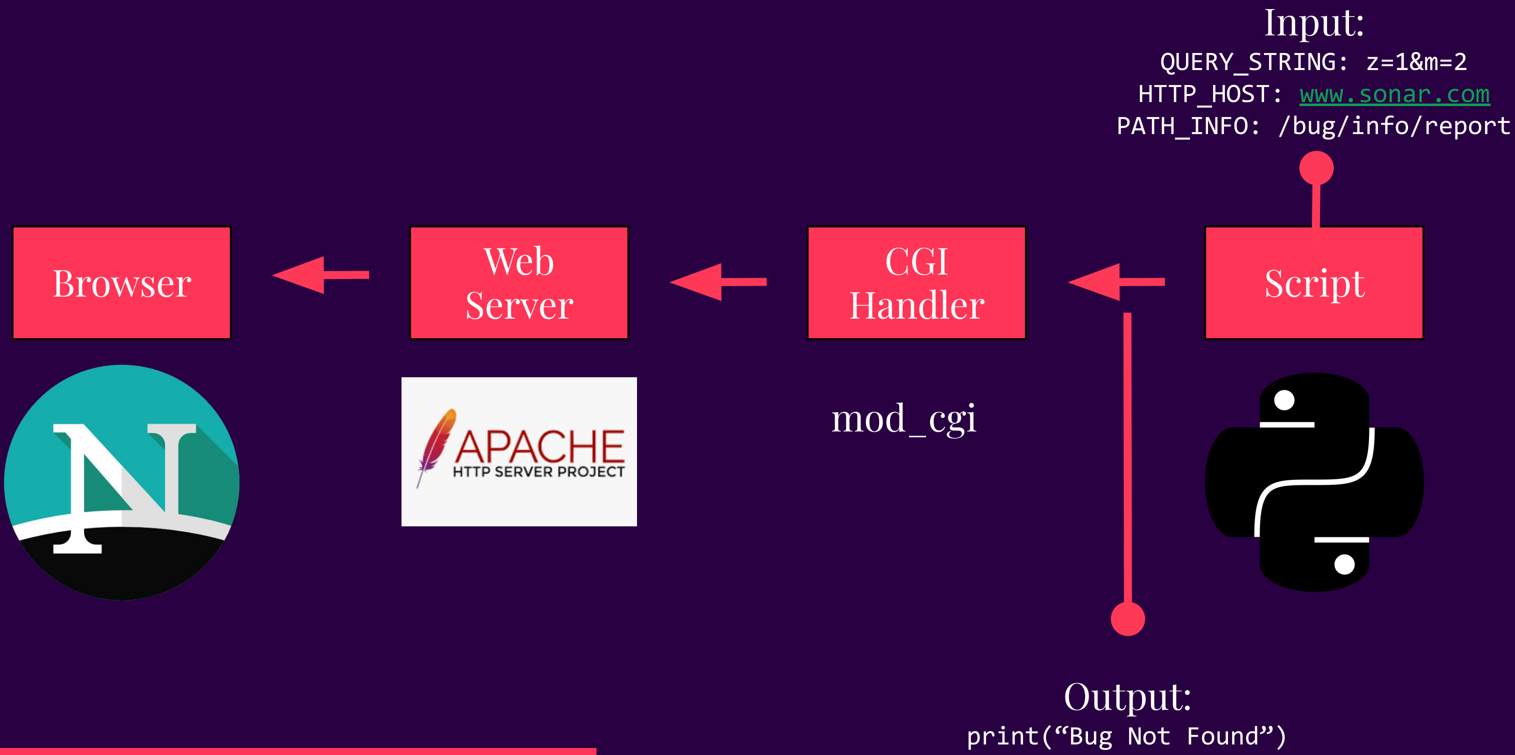
CGI [1991]



CGI: The Road to Dynamic Content



CGI: The Road to Dynamic Content



```
QUERY_STRING: z=1&m=2
HTTP_HOST: www.sonar.com
PATH_INFO: /bug/info/report
```

```
print(“Bug Not Found”)
```

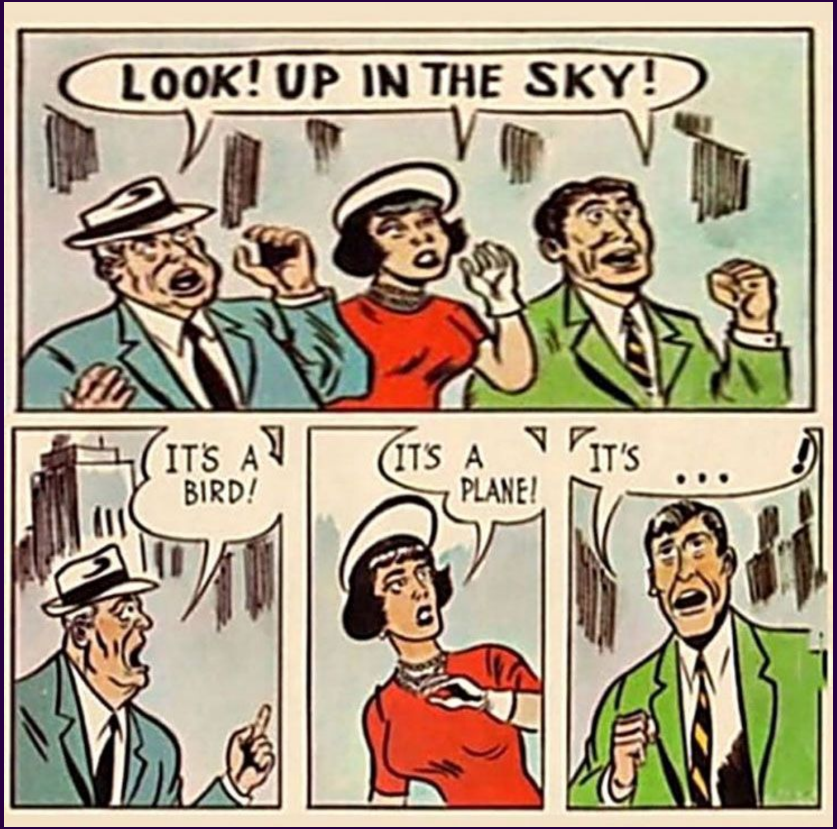
CGI: The Road to Dynamic Content

```
<form action =“/cgi-bin/login.py” method=“post”>
    <input type="text" name="firstname"  required>
    <input type="text" name="lastname"  required>
    <input type="email" name="email"  required>
    <input type="password" name="password"  required>
    <input type="submit" value="Login!">
</form>
```

1999

Pokémon™

Zope [1999]

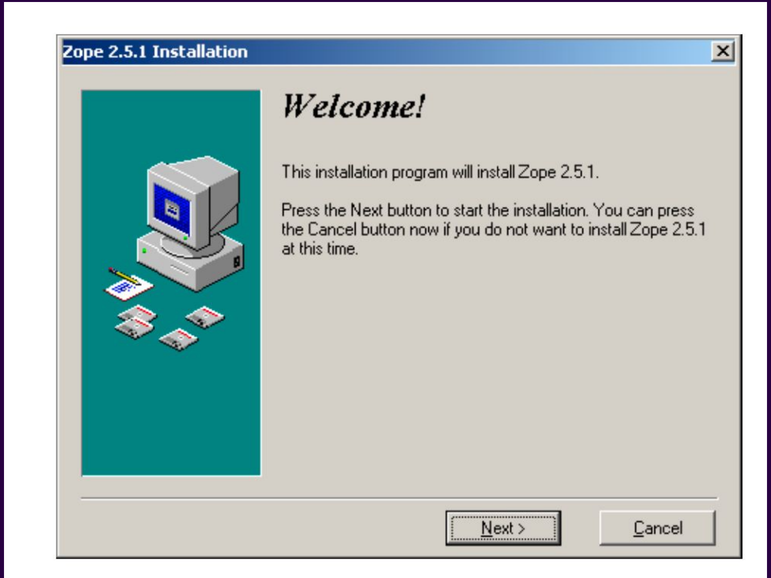


Zope: The Web CMS that did everything

Zope: The Web CMS that did everything

Zope is a **framework** that allows developers of varying skill levels to build **web applications**. – *The Zope Book*

All the following images are from "***The Zope Book***" by Amos Latteier and Michel Pelletier



Welcome!

This installation program will install Zope 2.5.1.

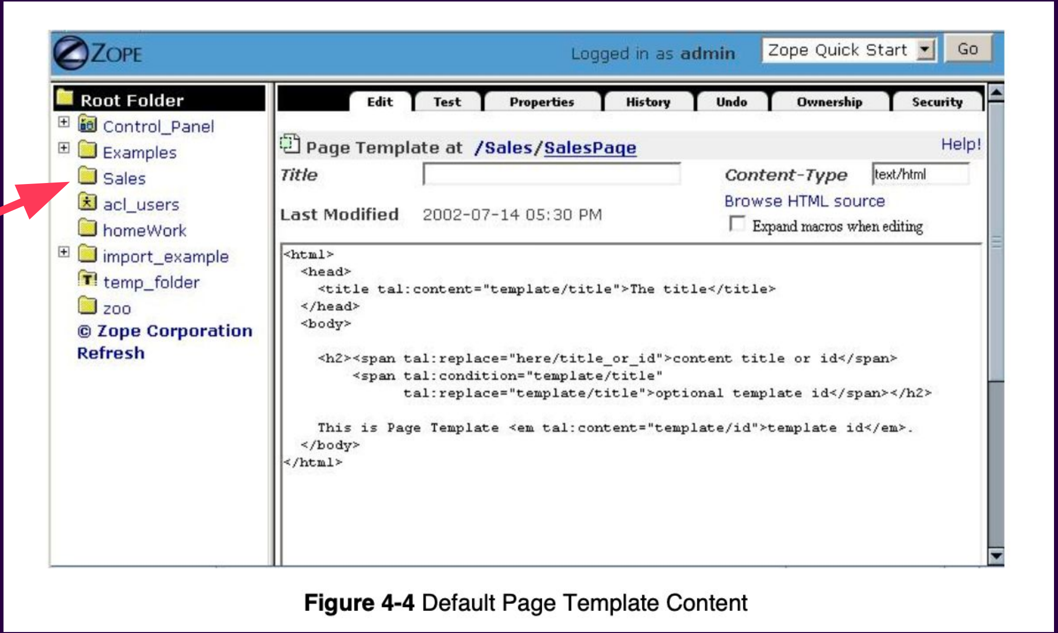
Press the Next button to start the installation. You can press the Cancel button now if you do not want to install Zope 2.5.1 at this time.

Figure 2-2 Beginning the installer

Zope 1 was very short lived. Most people started with Zope 2. Zope 1 was originally called Principia

Zope: The Web CMS that did everything

`http://localhost:8080/Sales/...`



Page Template at /Sales/SalesPage

Title

Content-Type text/html

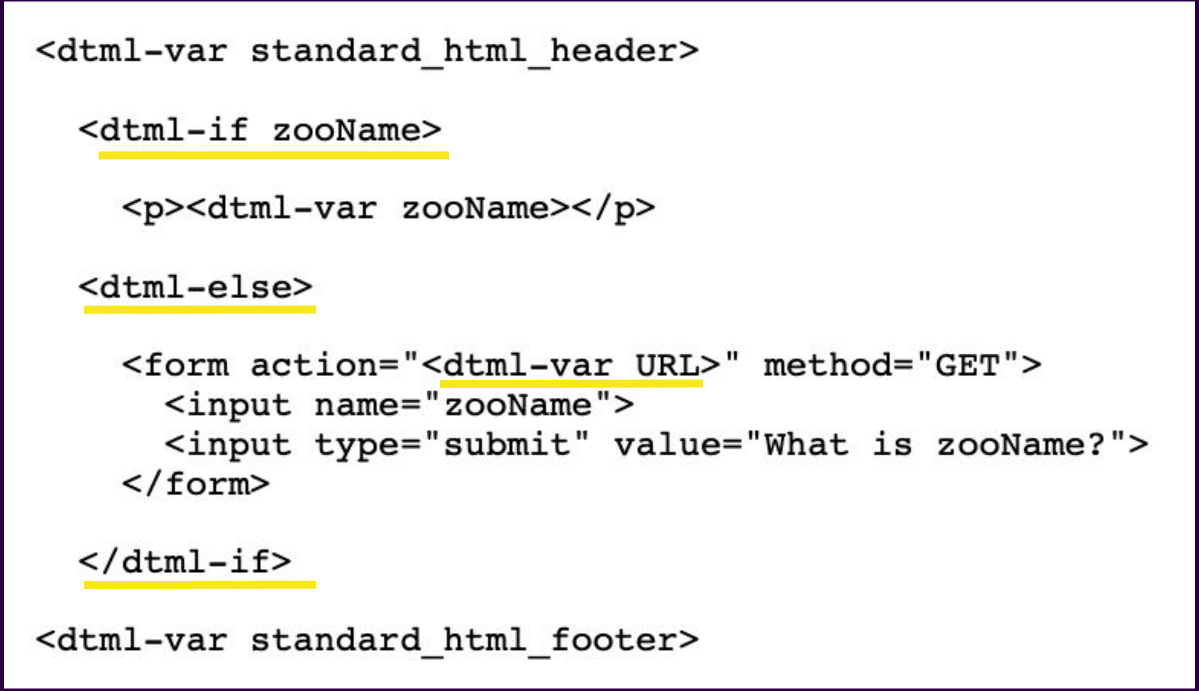
Last Modified 2002-07-14 05:30 PM

Expand macros when editing

```
<html>
  <head>
    <title tal:content="template/title">The title</title>
  </head>
  <body>

    <h2><span tal:replace="here/title_or_id">content title or id</span>
        <span tal:condition="template/title"
              tal:replace="template/title">optional template id</span></h2>

    This is Page Template <em tal:content="template/id">template id</em>.
  </body>
</html>
```

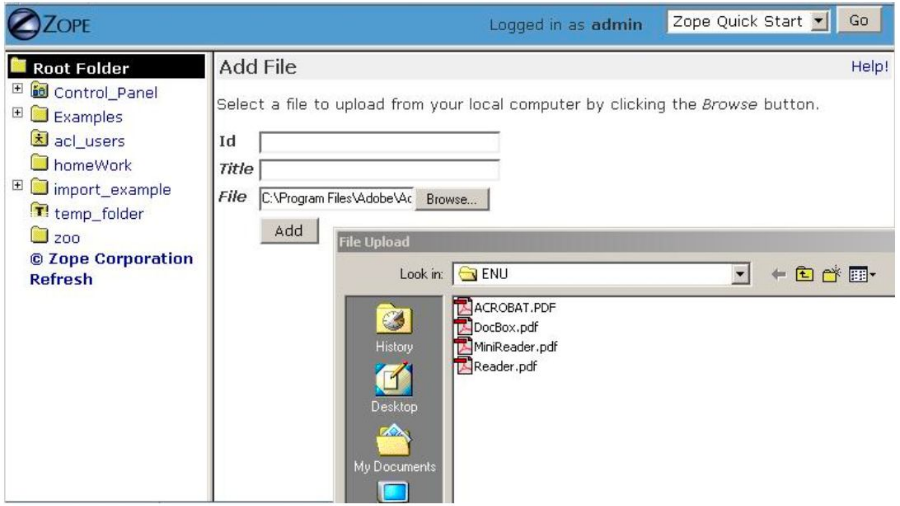
Figure 4-4 Default Page Template Content

Zope: The Web CMS that did everything



```
<dtml-var standard_html_header>

  <dtml-if zooName>

    <p><dtml-var zooName></p>

  <dtml-else>

    <form action="<dtml-var URL>" method="GET">
      <input name="zooName">
      <input type="submit" value="What is zooName?">
    </form>

  </dtml-if>

<dtml-var standard_html_footer>
```

Zope: DTML (Document Template Markup Language)

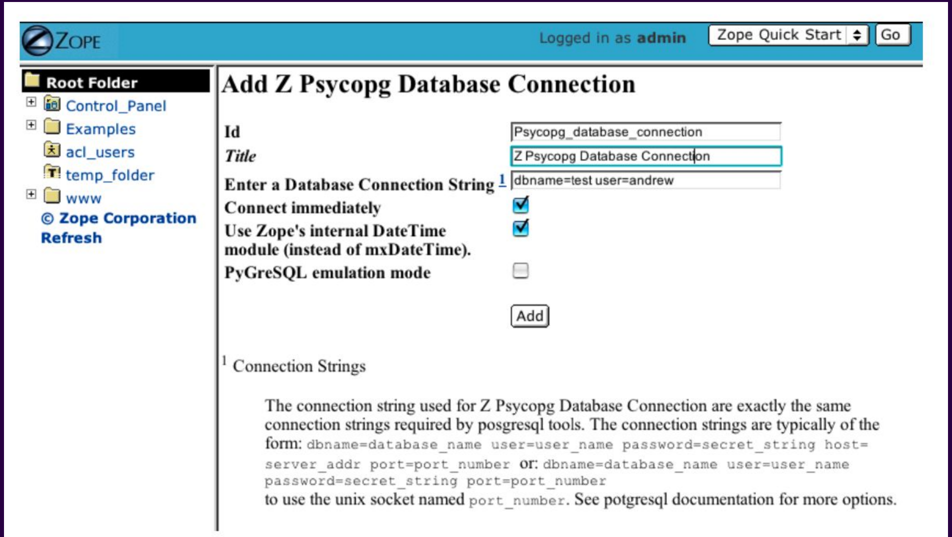


Select a file to upload from your local computer by clicking the *Browse* button.

Id	
Title	
File	C:\Program Files\Adobe\Ac

Figure 4-1 Adding a PDF File Object

Zope: The Web CMS that did everything



Add Z Psycopg Database Connection

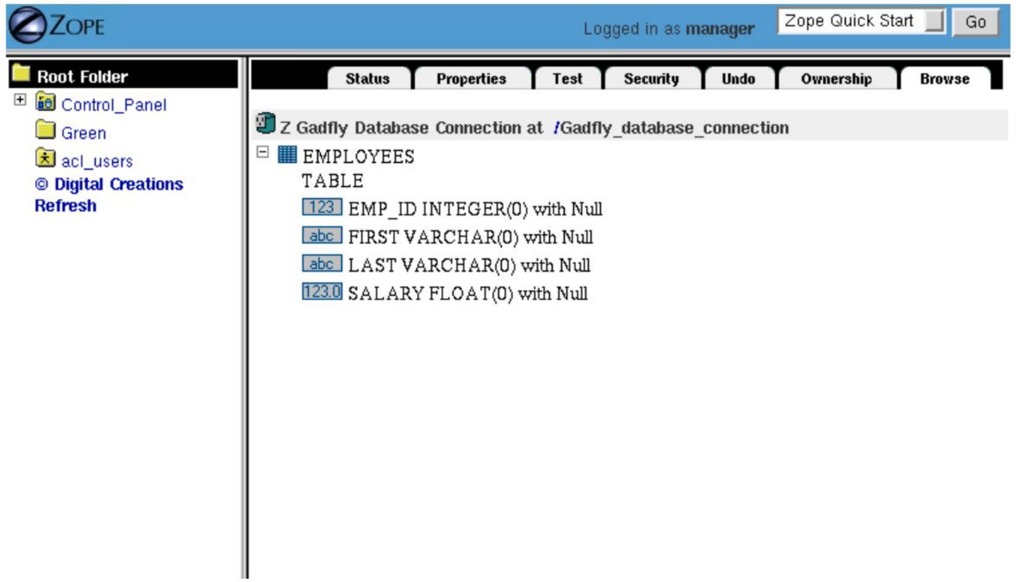
Id	Psycopg_database_connection
Title	Z Psycopg Database Connection
Enter a Database Connection String [1]	dbname=test user=andrew
Connect immediately	✓
Use Zope's internal DateTime module (instead of mxDateTime).	✓
PyGreSQL emulation mode	☐

[1] Connection Strings

The connection string used for Z Psycopg Database Connection are exactly the same connection strings required by posgresql tools. The connection strings are typically of the form: `dbname=database_name user=user_name password=secret_string host=server_addr port=port_number` or: `dbname=database_name user=user_name password=secret_string port=port_number` to use the unix socket named `port_number`. See potgresql documentation for more options.

Figure 17-1 PostgreSQL Database Connection

Zope: The Web CMS that did everything


Logged in as **manager**

Root Folder

Control_Panel

Green

acl_users

© Digital Creations

Refresh

Status | Properties | Test | Security | Undo | Ownership | Browse

Z Gadfly Database Connection at /Gadfly_database_connection

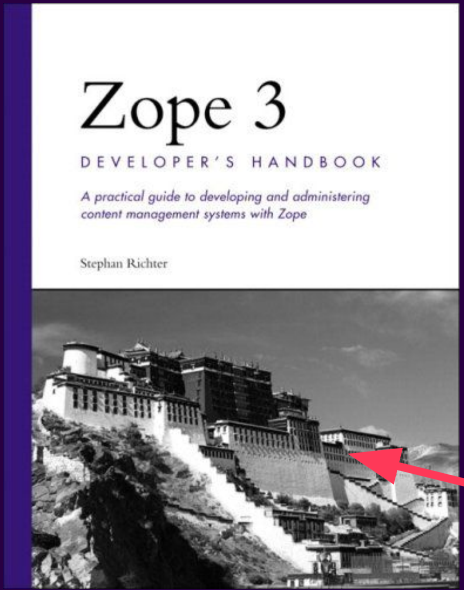
EMPLOYEES
TABLE
123 EMP_ID INTEGER(0) with Null
abc FIRST VARCHAR(0) with Null
abc LAST VARCHAR(0) with Null
123.0 SALARY FLOAT(0) with Null

Figure 17-3 Browsing the Database Connection

Zope: The Web CMS that did everything

Zope is called Zope because it is the **Z O**bject **P**ublishing **E**nvironment.

Everything was an object. Pages, Templates, Python Scripts, DTML. All of it was stored in **ZODB**.



Zope 3

DEVELOPER'S HANDBOOK

A practical guide to developing and administering content management systems with Zope

Stephan Richter

Potala Palace

Zope 2 was perhaps one of the most influential **open source** projects. Ever. But Zope 3 began the downfall of Zope

Zope: The Web CMS that did everything

All or Nothing. Zope did everything, and wasn’t modular. People started building things that were slightly better. It was **way** ahead of its time.

People wanted to use the goodies in from Zope 3 in Zope 2, so they used **five (3+2)**.

Zope: The Downfall (Zope 3 in 2004)

MEANWHILE...

Quixote [2000]

Quixote: The first modern? Web Framework?

```
from quixote.html import html_quote
from splat.web.util import get_bug_database
def _q_index (request):
   result = ["""\
       <html>
       <head><title>SPLAT! Bug Index</title></head>
       <body>
       <table>
         <tr>
           <th>bug id</th>
           <th>description</th>
         </tr>
       """]
   bug_db = get_bug_database()
   for bug in bug_db.get_all_bugs():
       if bug.status != bug.ST_RESOLVED:
           result.append("""\
               <tr>
                 <td>%s</td>
                 <td>%s</td>
               </tr>
               """ % (bug, html_quote(bug.description))
   result.append("""\
       </table>
       </body>
       </html>
       """)
   return "".join(result)
```

Quixote

Entry
Point

```python
from quixote.html import html_quote
from splat.web.util import get_bug_database
def _q_index (request):
   result = ["""\
       <html>
       <head><title>SPLAT! Bug Index</title></head>
       <body>
       <table>
         <tr>
           <th>bug id</th>
           <th>description</th>
         </tr>
       """]
   bug_db = get_bug_database()
   for bug in bug_db.get_all_bugs():
       if bug.status != bug.ST_RESOLVED:
           result.append("""\
               <tr>
                 <td>%s</td>
                 <td>%s</td>
               </tr>
               """ % (bug, html_quote(bug.description))
   result.append("""\
       </table>
       </body>
       </html>
       """)
   return "".join(result)
```

Quixote

Quixote: Python Templating Language (PTL)

Most Frameworks have HTML as a default. Quixote **inverted** that. JSX wasn't the first of its kind.

```
template header (title):
    """\
    <html>
    <head><title>SPLAT! - %s</title></head>
    <body>
    """ % html_quote(title)

template footer ():
    """\
    </table>
    </body>
    </html>
    """
```

Entry Point

```
template bug_row (bug):
    """\
      <tr>
        <td>%s</td>
        <td>%s</td>
      </tr>
    """ % (bug, html_quote(bug.description)

template _q_index (request, bug):
    header("Bug Index")
    """\
    <table>
      <tr>
        <th>bug id</th>
        <th>description</th>
      </tr>
    """
    bug_db = get_bug_database()
    for bug in bug_db.get_all_bugs():
        if bug.status != bug.ST_RESOLVED:
            bug_row(bug)
    "</table>\"
    footer()
```

Credit: Greg Ward

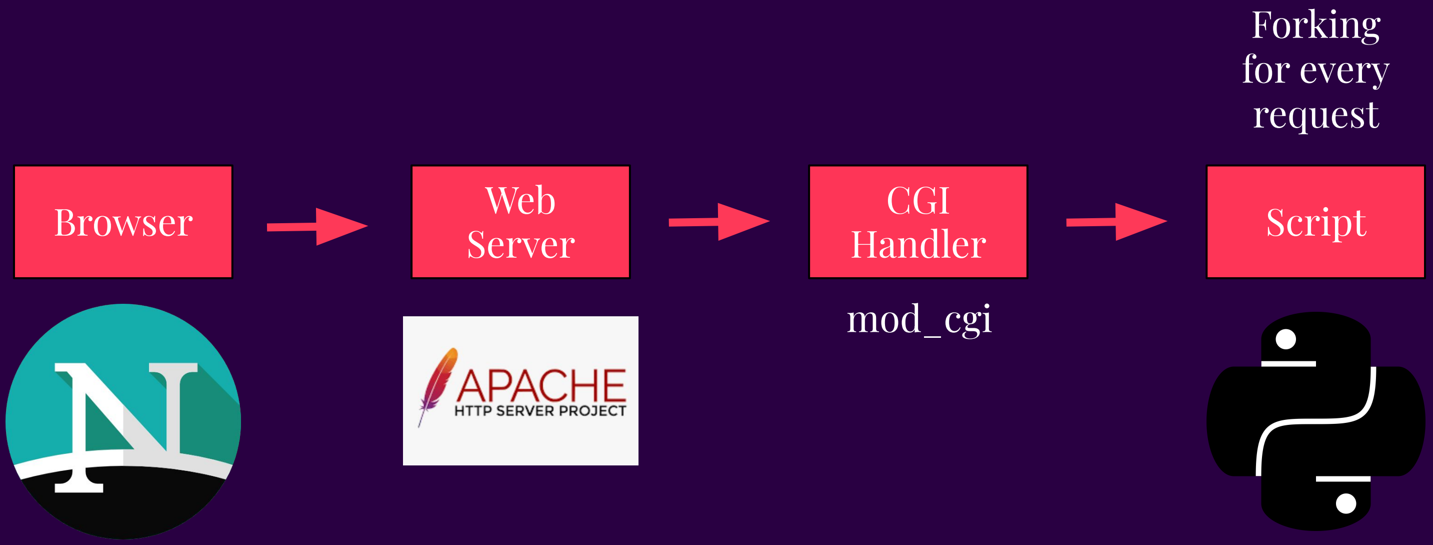
Quixote never really took off.

Quixote

MEANWHILE...

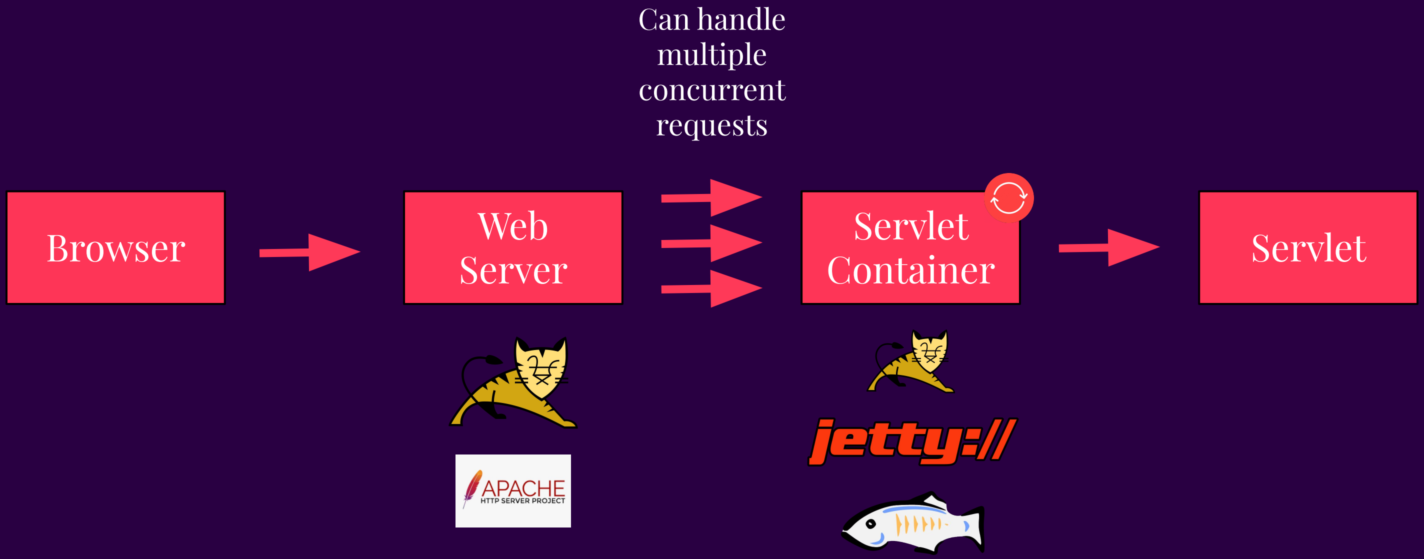
Webware for Python [2000]

Webware for Python: The Rise of JSP

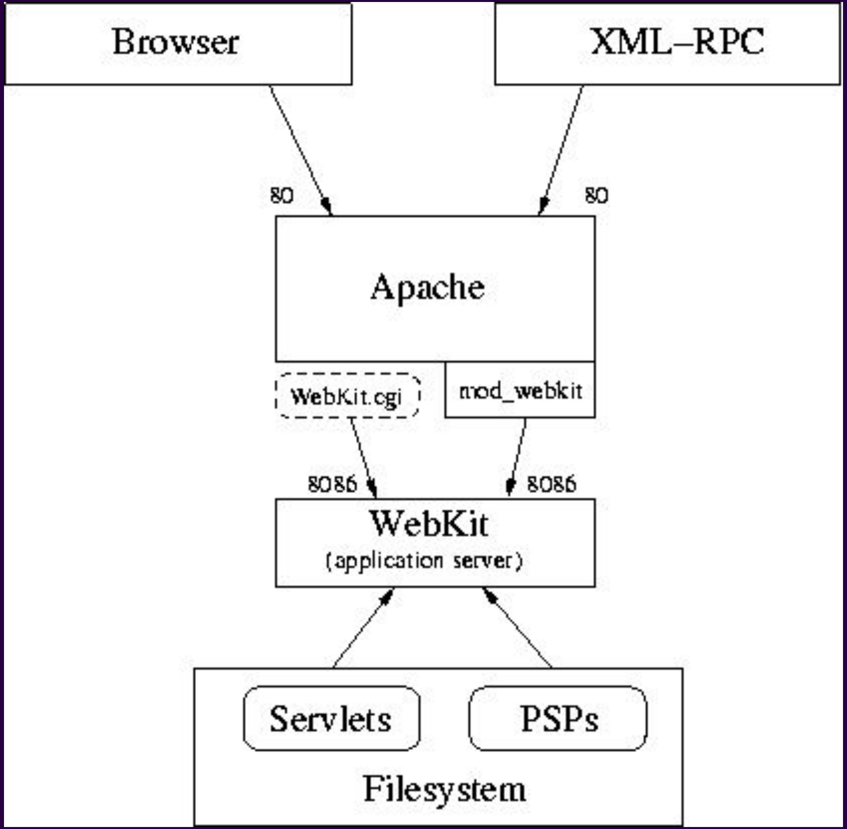


Webware for Python: Understanding CGI and Servlets

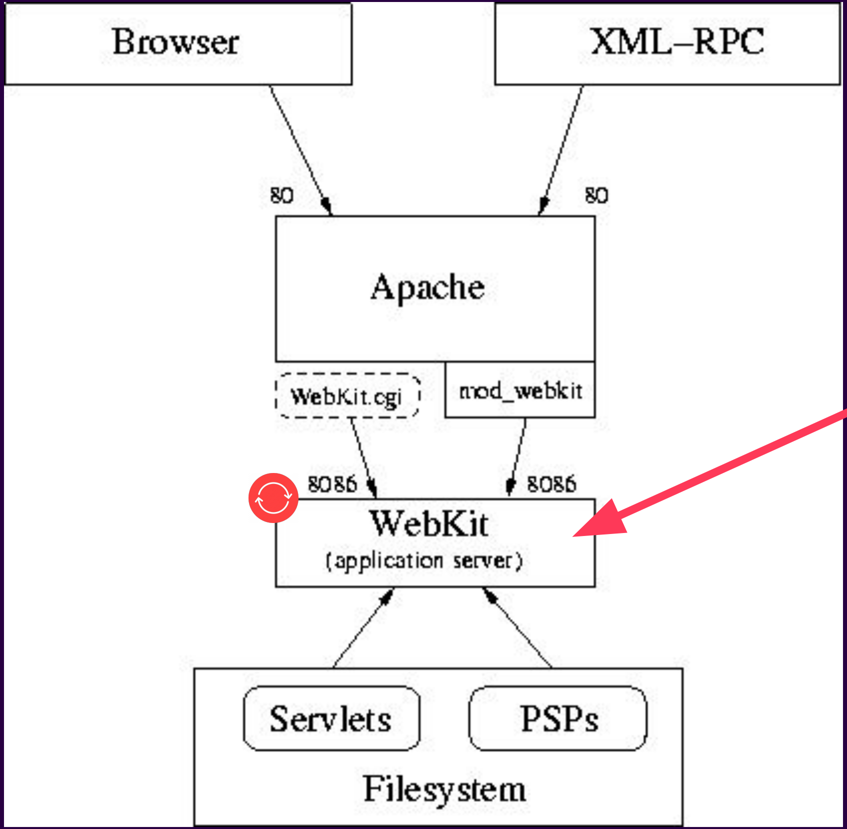
THIS IS FINE.


Can handle multiple concurrent requests
Browser
Web Server
Servlet Container
Servlet
APACHE HTTP SERVER PROJECT
jetty://

Webware for Python: Understanding Servlets



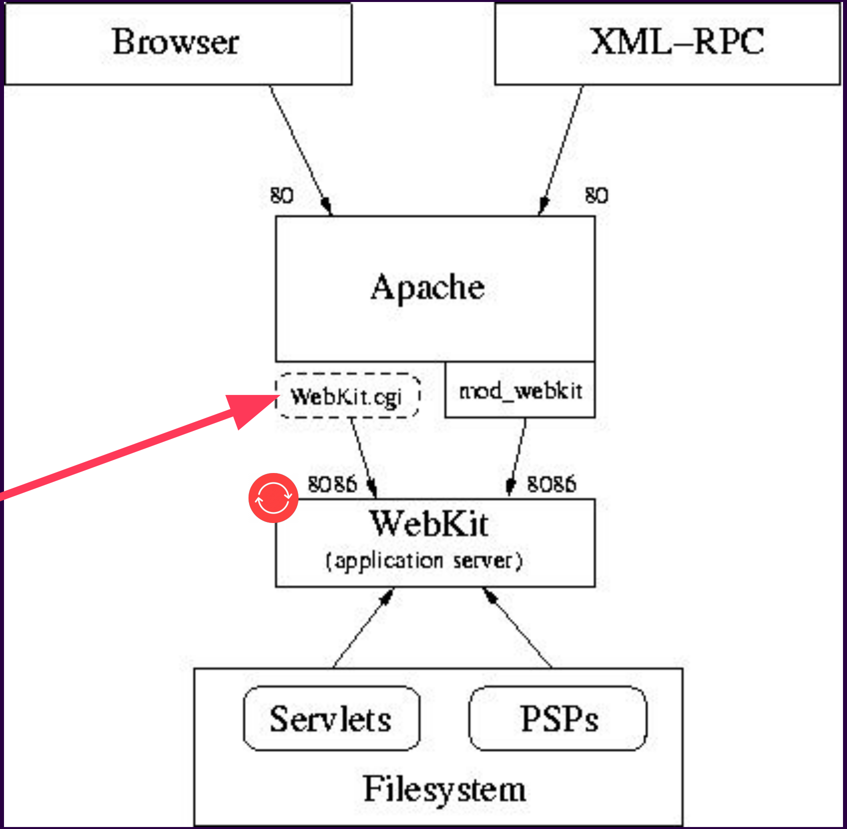
Webware for Python: The Java Stack but in Python



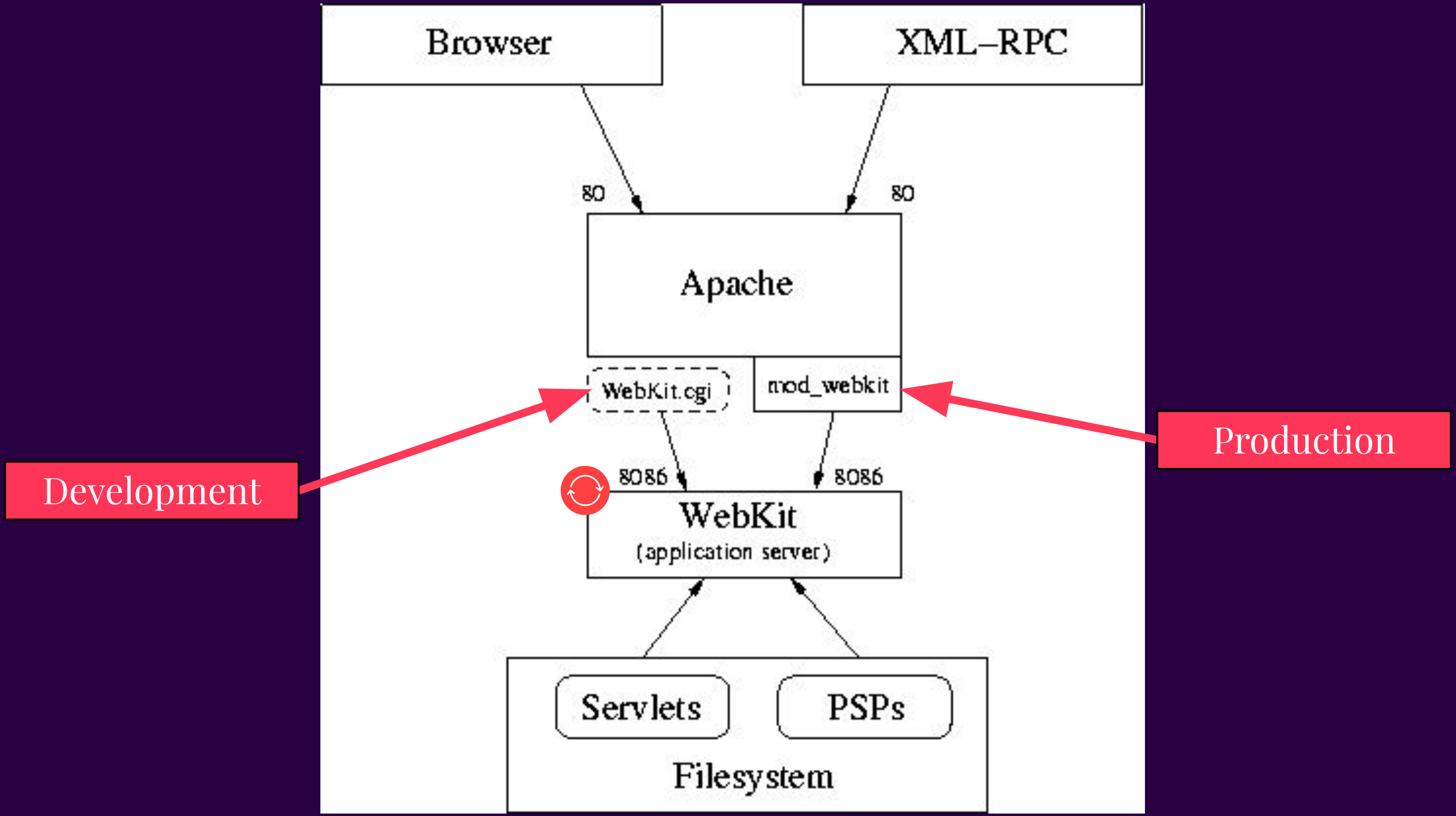
Multi-threaded application server

Webware for Python: The Java Stack but in Python

This would pass along the request to the Application Server

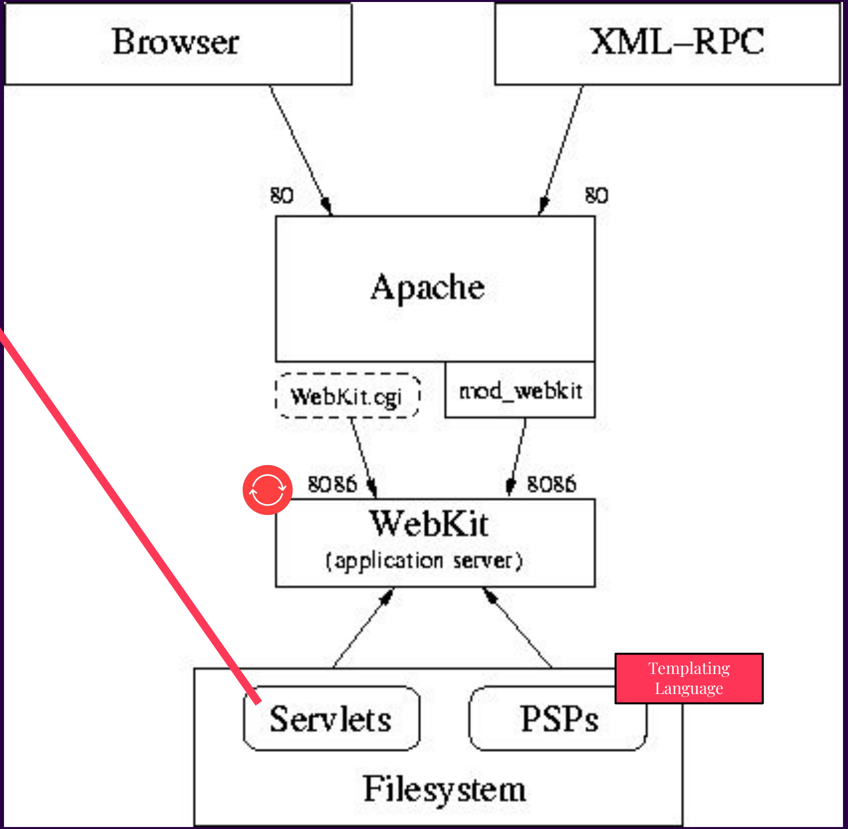


Webware for Python: The Java Stack but in Python

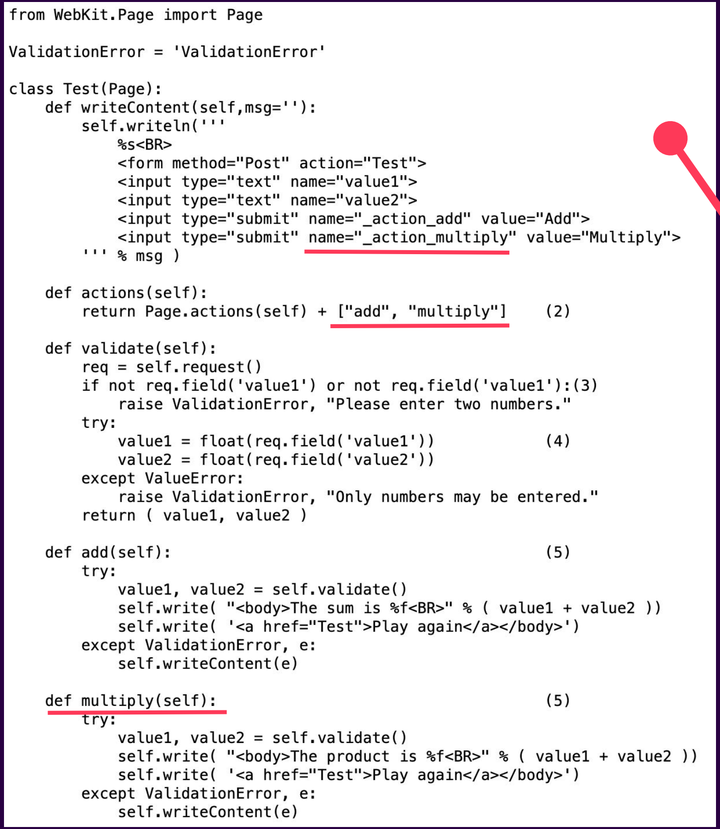


Webware for Python: The Java Stack but in Python

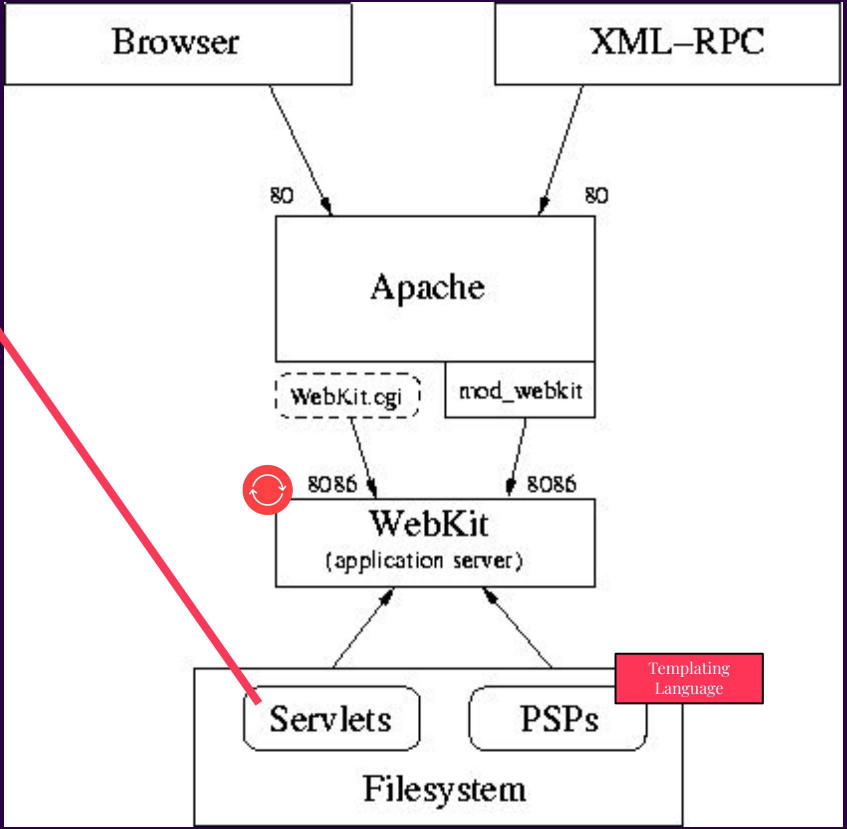
```
from WebKit.Page import Page

class HelloWorld(Page):
    def writeContent(self):
        self.writeln('Hello, world!')
```



Webware for Python: The Java Stack but in Python



```
from WebKit.Page import Page

ValidationError = 'ValidationError'

class Test(Page):
    def writeContent(self,msg=''):
        self.writeln('''
            %s<BR>
            <form method="Post" action="Test">
            <input type="text" name="value1">
            <input type="text" name="value2">
            <input type="submit" name="_action_add" value="Add">
            <input type="submit" name="_action_multiply" value="Multiply">
        ''' % msg )

    def actions(self):
        return Page.actions(self) + ["add", "multiply"]    (2)

    def validate(self):
        req = self.request()
        if not req.field('value1') or not req.field('value1'):(3)
            raise ValidationError, "Please enter two numbers."
        try:
            value1 = float(req.field('value1'))            (4)
            value2 = float(req.field('value2'))
        except ValueError:
            raise ValidationError, "Only numbers may be entered."
        return ( value1, value2 )

    def add(self):                                         (5)
        try:
            value1, value2 = self.validate()
            self.write( "<body>The sum is %f<BR>" % ( value1 + value2 ))
            self.write( '<a href="Test">Play again</a></body>')
        except ValidationError, e:
            self.writeContent(e)

    def multiply(self):                                    (5)
        try:
            value1, value2 = self.validate()
            self.write( "<body>The product is %f<BR>" % ( value1 + value2 ))
            self.write( '<a href="Test">Play again</a></body>')
        except ValidationError, e:
            self.writeContent(e)
```



Webware for Python: The Java Stack but in Python

Files Overview

```
Videos/
    Middle/
        generate*
        create*
        insert*
        Videos.mkmodel/
            Classes.csv
            Samples.csv
            Settings.config
        GeneratedPy/
            GenVideo.py, GenMovie.py, ...
        GeneratedSQL/
            Create.sql
            InsertSamples.sql
            Info.text
        Video.py
        Movie.py
        ...
    Command/
        main.py
```

Setting up Middlekit

```
# Gain access to the Middle package
import os, sys
sys.path.insert(1, os.path.abspath(os.pardir))

from datetime import date
from MiddleKit.Run.MySQLObjectStore import MySQLObjectStore
from Middle.Movie import Movie

def main():
    # Set up the store
    store = MySQLObjectStore(user='user', passwd='password')
    store.readModelFileNamed('../Middle/Videos')

    movie = Movie()
    movie.setTitle('The Terminator')
    movie.setYear(1984)
    movie.setRating('r')
    store.addObject(movie)
    store.saveChanges()

if __name__=='__main__':
    main()
```

Querying

```
videos = store.fetchObjectsOfClass('Video')
# Get all videos that start with 'A':
videos = [video for video in videos if video.title().upper().startswith('A')]
```

Webware for Python: Middlekit (ORM)

```
<psp:file>
  # Since this is at the module level, _log is only defined once for this file
  import logging
  _log = logging.getLogger( __name__ )
</psp:file>
<html>
  <% _log.debug('Okay, I’ve been called.') %>
  <p>Write stuff here.</p>
</html>
```

Python files inside HTML

```
<psp:class>
 def writeNavBar(self):
    for uri, title in self.menuPages():
      self.write( "<a href="%s">%s</a>" % (uri, title) )
</psp:class>
```

Declaring Python classes in HTML

```
<% for i in range(5):
      res.write("<b>This is number" + str(i) + "</b><br>") %>
```

Good old for loops in HTML!

Webware for Python: PSP

Webware for Python still is **actively maintained**.

Interlude

Web Server

Caddy®
THE ULTIMATE SERVER

NGINX®

APACHE
HTTP SERVER PROJECT

Getting requests and handling static content. Also handles SSL

Application Server

uvicorn

gunicorn

uWSGI

Forwards requests, and handles instances of the Framework applications

Web Framework

FastAPI

django

Houses the Application Code

The Modern Request Pipeline: Things are Coalescing

A FEW
MOMENTS
LATER

WSGI [2001]

WI-Z-GI

Web Server Gateway Interface:
Final version in 2003

WSGI: A Successor to CGI for Python

This is the callable that is passed into the
WSGI server.

```python
def simple_app(environ, start_response):
    status = '200 OK'
    response_headers = [('Content-type', 'text/plain')]
    start_response(status, response_headers)
    return []
```

WSGI: A Successor to CGI for Python

Similar to CGI, this passes information like `REQUEST_METHOD`, `QUERY_STRING`

```
def simple_app(environ, start_response):
    status = '200 OK'
    response_headers = [('Content-type', 'text/plain')]
    start_response(status, response_headers)
    return []
```

WSGI: A Successor to CGI for Python

The callable that is used to create the response.

```
def simple_app(environ, start_response):
    status = '200 OK'
    response_headers = [('Content-type', 'text/plain')]
    start_response(status, response_headers)
    return []
```

WSGI: A Successor to CGI for Python

```python
def simple_app(environ, start_response):
    """Simplest possible application object"""
    status = '200 OK'
    response_headers = [('Content-type', 'text/plain')]
    response = your_view_function(environ)
    start_response(status, response_headers)
    return [response]
```

This function **is provided** by the WSGI server itself. So a server like gunicorn will have this available for you.

WSGI: A Successor to CGI for Python

simple_app.py

```python
def hello_view(environ, start_response):
    """
    A view function that returns "Hello World".
    """
    status = '200 OK'
    headers = [('Content-type', 'text/plain')]
    start_response(status, headers)
    return [b"Hello World"]

def goodbye_view(environ, start_response):
    """
    A view function that returns "Goodbye World".
    """
    status = '200 OK'
    headers = [('Content-type', 'text/plain')]
    start_response(status, headers)
    return [b"Goodbye World"]
```

```python
def application(environ, start_response):
    """
    The WSGI callable. It routes requests based on the URL path.
    """
    path = environ.get('PATH_INFO', '')

    if path == '/hello':
        return hello_view(environ, start_response)
    elif path == '/goodbye':
        return goodbye_view(environ, start_response)
    else:
        status = '404 Not Found'
        headers = [('Content-type', 'text/plain')]
        start_response(status, headers)
        return [b"404 - Not Found"]
```

```
~ -> gunicorn simpleapp:application
```

WSGI: A Successor to CGI for Python

WSGI: The Inevitable Rise of Python Web Frameworks

CherryPy [2002]

CherryPy is in between a **compiler** and an **application server**.

CherryPy

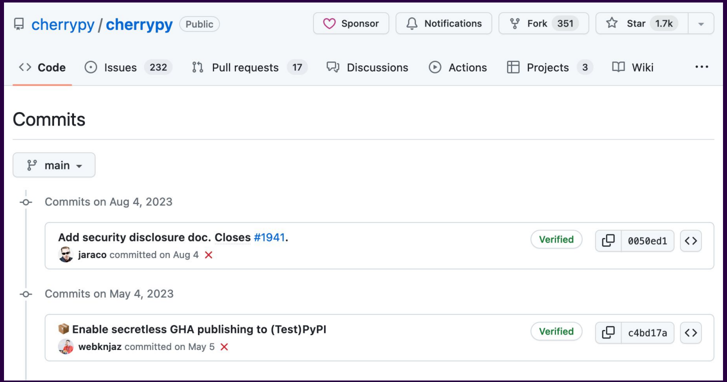
```
Hello.cpy
```

```
CherryClass Root:
mask:
    def index(self, name="you"):
        <html><body>
            Hello, <b py-eval="name"></b> !
            <form py-attr="request.base" action="" method="get">
                Enter your name: <input name=name type=text><br>
                <input type=submit value=OK>
            </form>
        </body></html>
```

This mask allowed you to use CherryPy's templating language.

```
~ -> python ../cherrypy.py Hello.cpy
```

CherryPy: Remind you of JSX?


cherrypy / cherrypy Public

Sponsor · Notifications · Fork 351 · Star 1.7k

Code · Issues 232 · Pull requests 17 · Discussions · Actions · Projects 3 · Wiki

Commits

main

Commits on Aug 4, 2023

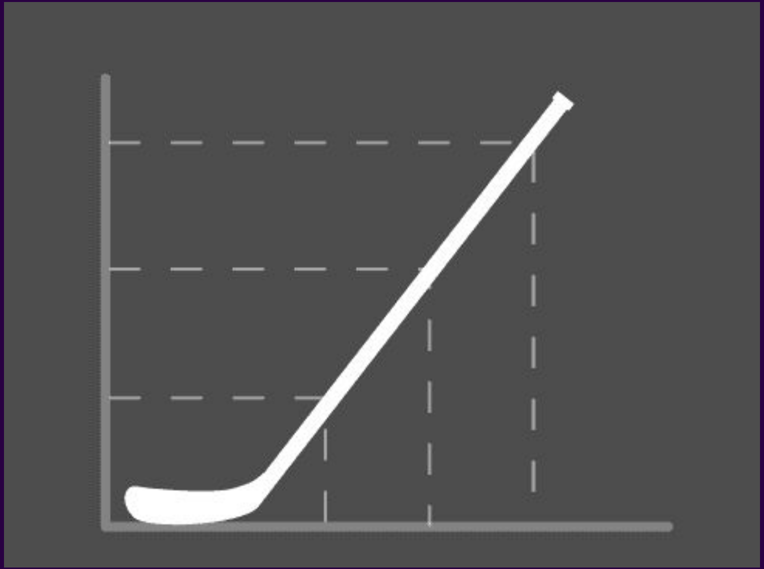
Add security disclosure doc. Closes #1941.
jaraco committed on Aug 4 — Verified — 0050ed1

Commits on May 4, 2023

📦 Enable secretless GHA publishing to (Test)PyPI
webknjaz committed on May 5 — Verified — c4bd17a

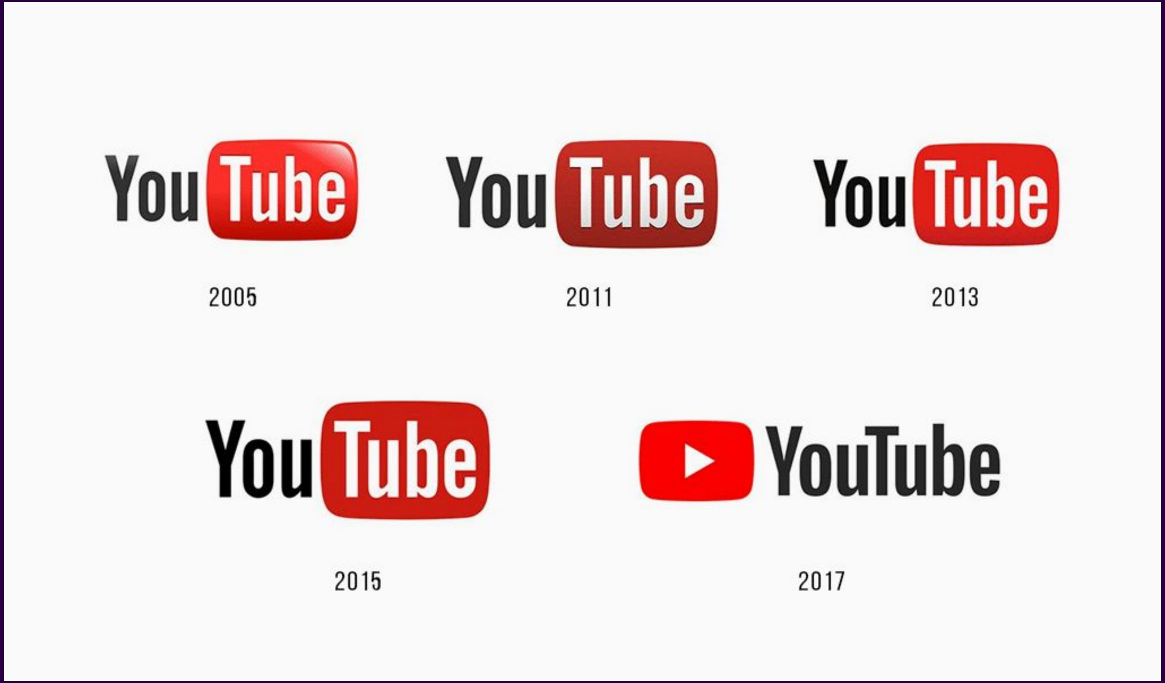
CherryPy is still alive, and moved to WSGI in 2005. It is not compatible with ASGI.

CherryPy: Remind you of JSX?



After 2005, it was never the same again.

2005



TurboGears [2005]

Integrated Full-Stack Framework: ActiveRecord, ActiveView and ActiveController

RESTful Development

Convenience over Configuration: Sensible Defaults

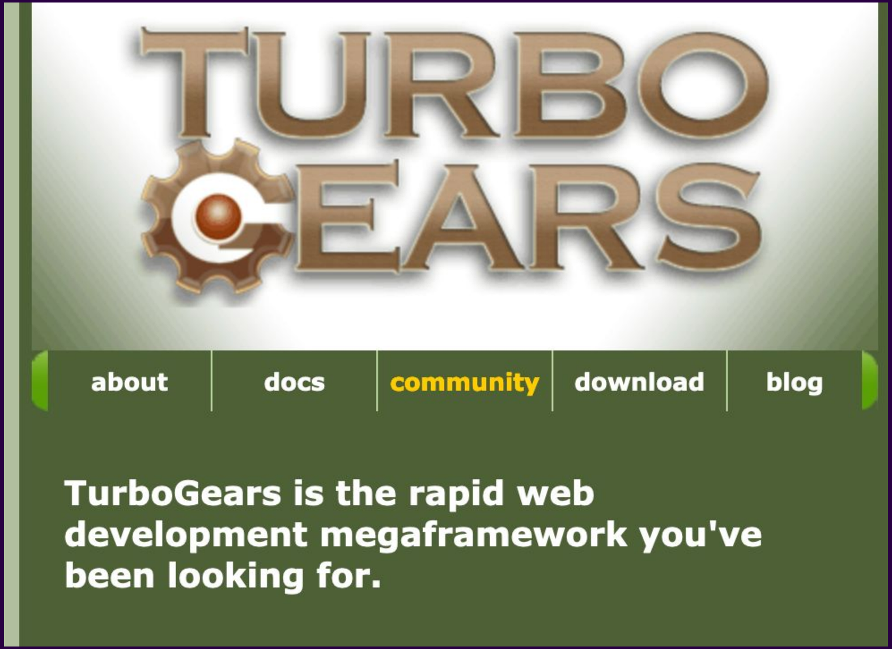
Scaffolding tools and Generators

Great Docs: More and more tools live and die by the quality of docs

Database Migrations

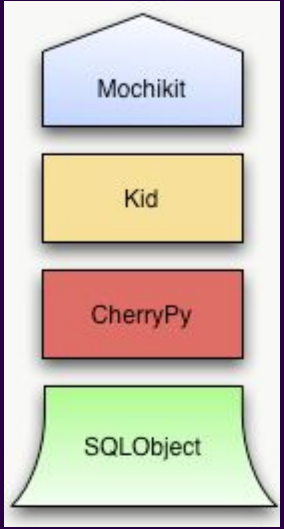
Clear “Best Practices”: Good new newcomers

TurboGears: Rails was on the rise



TurboGears is the rapid web development megaframework you've been looking for.

TurboGears

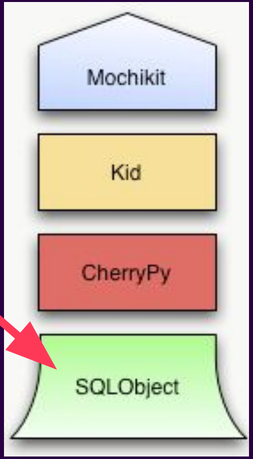


TurboGears: The Stack

```python
from sqlobject import *
from datetime import datetime


class Person(SQLObject):

    firstName = StringCol(length=100)
    middleInitial = StringCol(length=1, default=None)
    lastName = StringCol(length=100)
    lastContact = DateTimeCol(default=datetime.now)
```

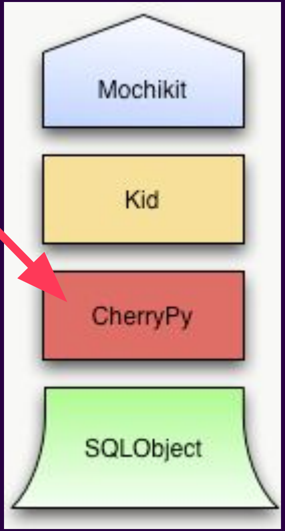
```
>>> p = Person(firstName="John", lastName="Doe")
>>> p
<Person 1 firstName='John' middleInitial=None
lastName='Doe' lastContact='datetime.datetime...)'>
>>> p.lastContact
datetime.datetime(2005, 9, 16, 9, 28, 7)
>>> p.firstName
'John'
>>> p.middleInitial = 'Q'
>>> p.middleInitial
'Q'
>>> p2 = Person.get(1)
>>> p is p2
True
```

The ORM: Uses an ActiveRecord Pattern. This meant that **each record** could perform **CRUD operations** because the objects containers both **data and behaviour**.



TurboGears: The Stack

Controller and View Layer



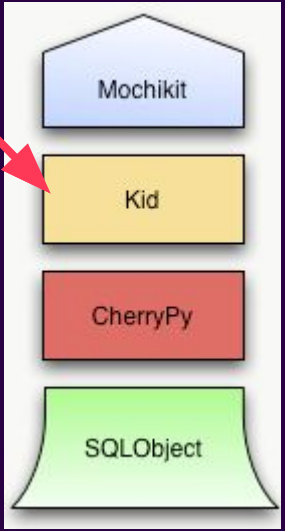
```
import cherrypy

class MyRoot:

    @cherrypy.expose()
    def index(self, who="World"):
        return "Hello, %s!" % (who)
```

TurboGears: The Stack

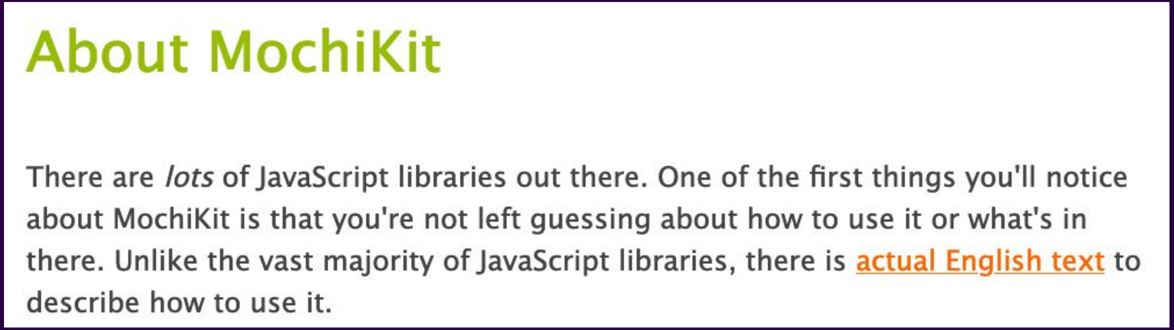
```
print "<table>"
for person in people:
    print "<tr>"
    print "<td>%s</td>" % (person.name)
    print "</tr>"
print "</table>"
```

```
<table>
    <tr py:for="person in people">
        <td><span py:content="person.name">Kevin Bacon</span></td>
    </tr>
</table>
```

The Templating Engine

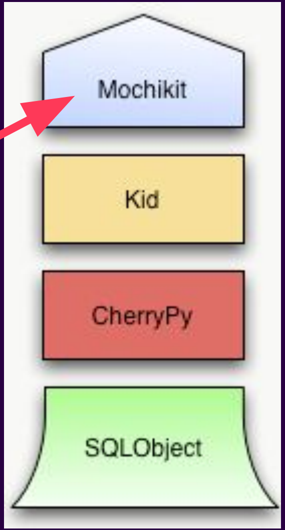


TurboGears: The Stack


About MochiKit

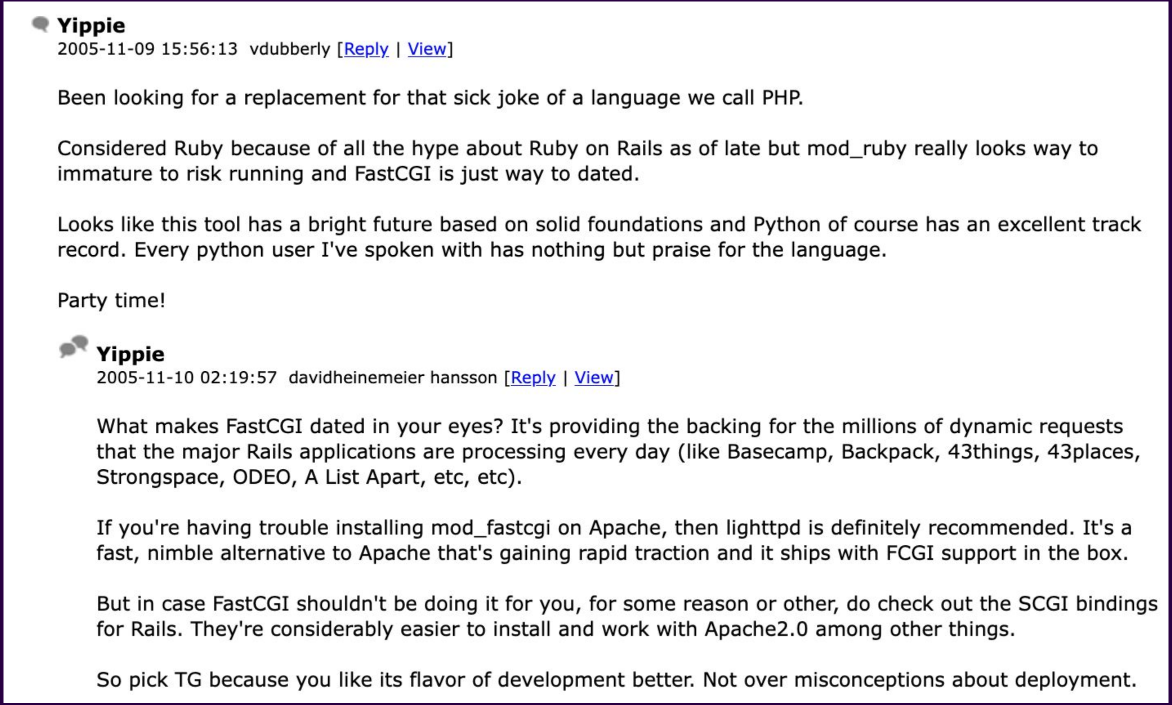
There are *lots* of JavaScript libraries out there. One of the first things you'll notice about MochiKit is that you're not left guessing about how to use it or what's in there. Unlike the vast majority of JavaScript libraries, there is actual English text to describe how to use it.

The JavaScript Library that
helps with AJAX



TurboGears: The Stack

TurboGears initially used CherryPy’s server, which was HTTP 1.1 compliant. But later on moved onto WSGI when it gained traction as did many other frameworks.



Yippie
2005-11-09 15:56:13 vdubberly [Reply | View]

Been looking for a replacement for that sick joke of a language we call PHP.

Considered Ruby because of all the hype about Ruby on Rails as of late but mod_ruby really looks way to immature to risk running and FastCGI is just way to dated.

Looks like this tool has a bright future based on solid foundations and Python of course has an excellent track record. Every python user I've spoken with has nothing but praise for the language.

Party time!

Yippie
2005-11-10 02:19:57 davidheinemeier hansson [Reply | View]

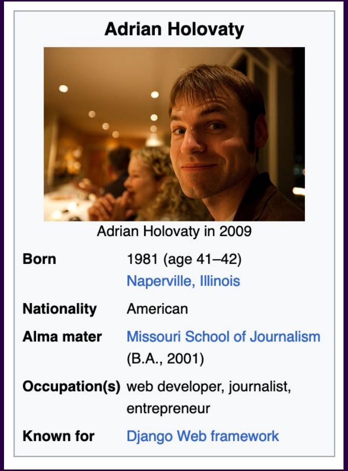
What makes FastCGI dated in your eyes? It's providing the backing for the millions of dynamic requests that the major Rails applications are processing every day (like Basecamp, Backpack, 43things, 43places, Strongspace, ODEO, A List Apart, etc, etc).

If you're having trouble installing mod_fastcgi on Apache, then lighttpd is definitely recommended. It's a fast, nimble alternative to Apache that's gaining rapid traction and it ships with FCGI support in the box.

But in case FastCGI shouldn't be doing it for you, for some reason or other, do check out the SCGI bindings for Rails. They're considerably easier to install and work with Apache2.0 among other things.

So pick TG because you like its flavor of development better. Not over misconceptions about deployment.

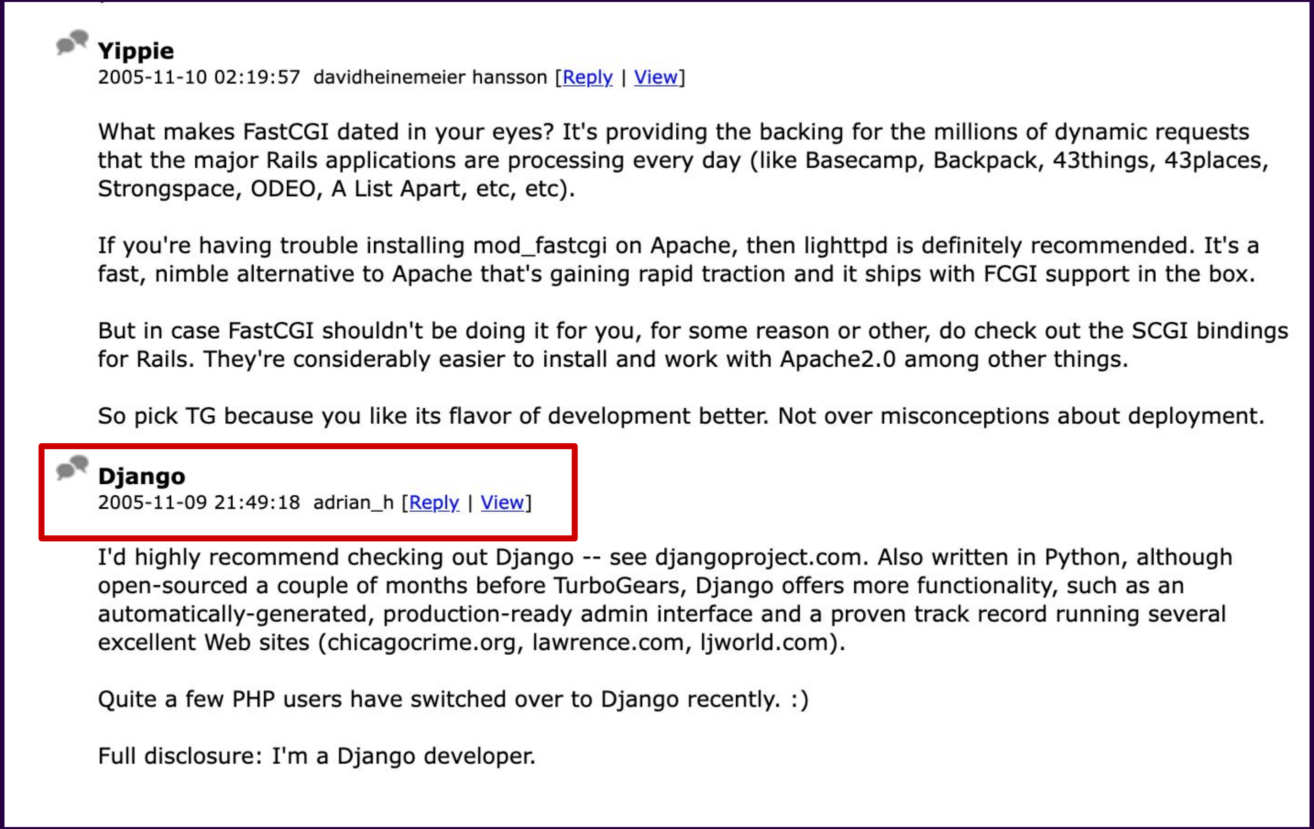
TurboGears: The Comments!



Adrian Holovaty

Adrian Holovaty in 2009

Born	1981 (age 41–42) Naperville, Illinois
Nationality	American
Alma mater	Missouri School of Journalism (B.A., 2001)
Occupation(s)	web developer, journalist, entrepreneur
Known for	Django Web framework



Yippie
2005-11-10 02:19:57 davidheinemeier hansson [Reply | View]

What makes FastCGI dated in your eyes? It's providing the backing for the millions of dynamic requests that the major Rails applications are processing every day (like Basecamp, Backpack, 43things, 43places, Strongspace, ODEO, A List Apart, etc, etc).

If you're having trouble installing mod_fastcgi on Apache, then lighttpd is definitely recommended. It's a fast, nimble alternative to Apache that's gaining rapid traction and it ships with FCGI support in the box.

But in case FastCGI shouldn't be doing it for you, for some reason or other, do check out the SCGI bindings for Rails. They're considerably easier to install and work with Apache2.0 among other things.

So pick TG because you like its flavor of development better. Not over misconceptions about deployment.

Django
2005-11-09 21:49:18 adrian_h [Reply | View]

I'd highly recommend checking out Django -- see djangoproject.com. Also written in Python, although open-sourced a couple of months before TurboGears, Django offers more functionality, such as an automatically-generated, production-ready admin interface and a proven track record running several excellent Web sites (chicagocrime.org, lawrence.com, ljworld.com).

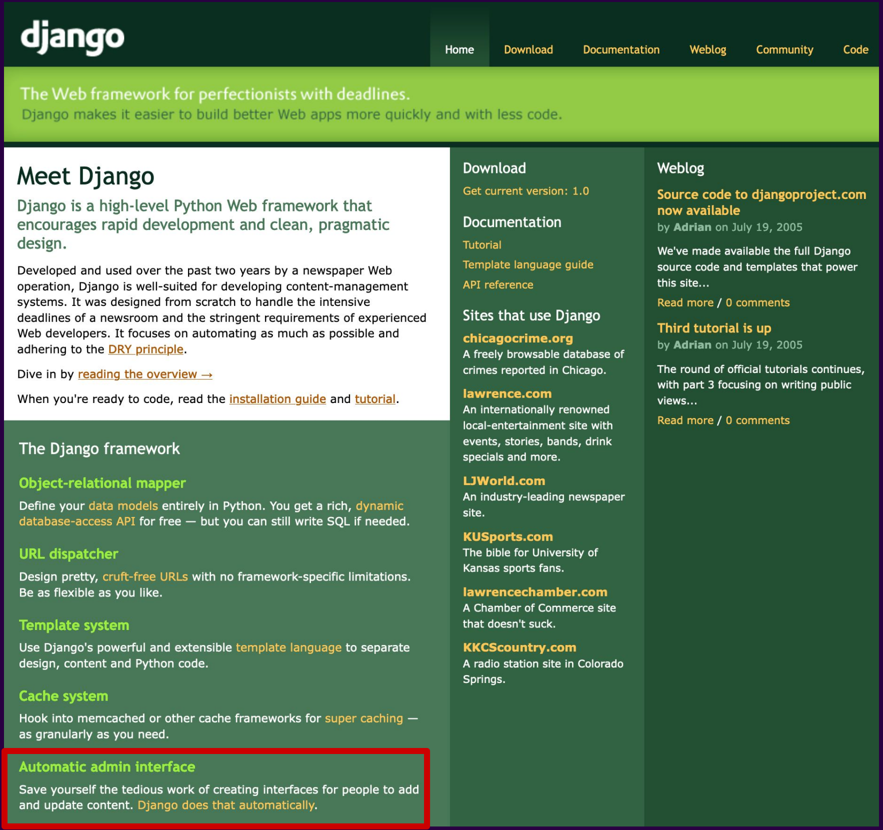
Quite a few PHP users have switched over to Django recently. :)

Full disclosure: I'm a Django developer.

A New Framework

ANOTHER ONE

Django [2005]


django

Home Download Documentation Weblog Community Code

The Web framework for perfectionists with deadlines.
Django makes it easier to build better Web apps more quickly and with less code.

Meet Django

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design.

Developed and used over the past two years by a newspaper Web operation, Django is well-suited for developing content-management systems. It was designed from scratch to handle the intensive deadlines of a newsroom and the stringent requirements of experienced Web developers. It focuses on automating as much as possible and adhering to the DRY principle.

Dive in by reading the overview →

When you're ready to code, read the installation guide and tutorial.

The Django framework

Object-relational mapper
Define your data models entirely in Python. You get a rich, dynamic database-access API for free — but you can still write SQL if needed.

URL dispatcher
Design pretty, cruft-free URLs with no framework-specific limitations. Be as flexible as you like.

Template system
Use Django's powerful and extensible template language to separate design, content and Python code.

Cache system
Hook into memcached or other cache frameworks for super caching — as granularly as you need.

Automatic admin interface
Save yourself the tedious work of creating interfaces for people to add and update content. Django does that automatically.

Download
Get current version: 1.0

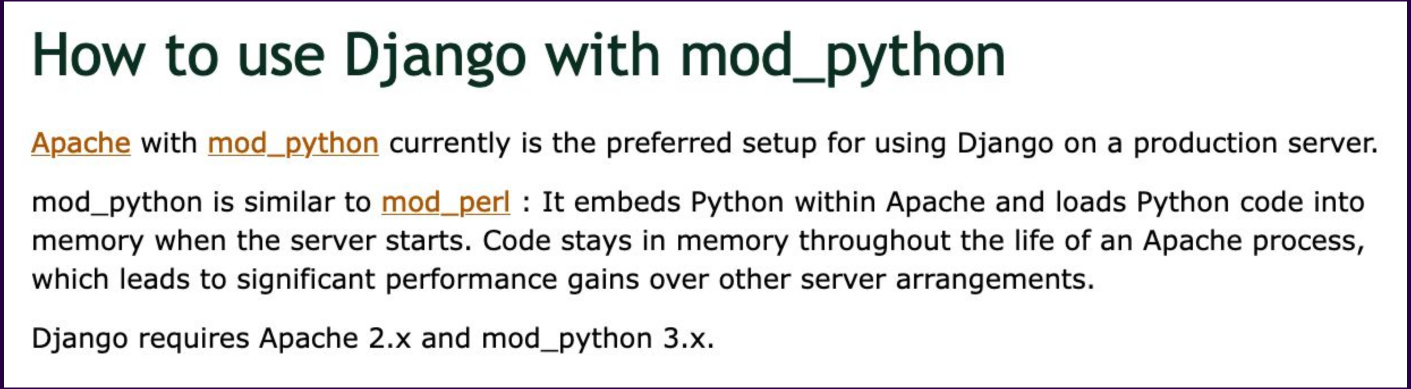
Documentation
Tutorial
Template language guide
API reference

Sites that use Django
chicagocrime.org
A freely browsable database of crimes reported in Chicago.
lawrence.com
An internationally renowned local-entertainment site with events, stories, bands, drink specials and more.
LJWorld.com
An industry-leading newspaper site.
KUSports.com
The bible for University of Kansas sports fans.
lawrencechamber.com
A Chamber of Commerce site that doesn't suck.
KKCScountry.com
A radio station site in Colorado Springs.

Weblog
Source code to djangoproject.com now available
by **Adrian** on July 19, 2005
We've made available the full Django source code and templates that power this site...
Read more / 0 comments

Third tutorial is up
by **Adrian** on July 19, 2005
The round of official tutorials continues, with part 3 focusing on writing public views...
Read more / 0 comments

Django: Django all the way down!



How to use Django with mod_python

Apache with mod_python currently is the preferred setup for using Django on a production server.

mod_python is similar to mod_perl : It embeds Python within Apache and loads Python code into memory when the server starts. Code stays in memory throughout the life of an Apache process, which leads to significant performance gains over other server arrangements.

Django requires Apache 2.x and mod_python 3.x.

Django: It didn’t use WSGI to start with urls.py

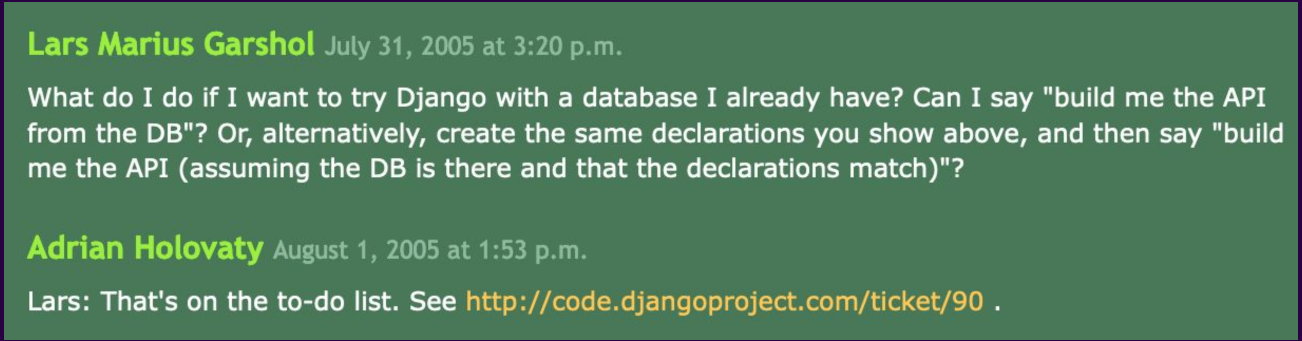
```
from django.conf.urls.defaults import *

urlpatterns = patterns('',
    (r'^/articles/(?P<year>\d{4})/$', 'myproject.news.views.year_archive'),
    (r'^/articles/(?P<year>\d{4})/(?P<month>\d{2})/$', 'myproject.news.views.month_archive'),
    (r'^/articles/(?P<year>\d{4})/(?P<month>\d{2})/(?P<article_id>\d+)/$', 'myproject.news.views.article_detail'),
)
```

views.py

```
def article_detail(request, year, month, article_id):
    # Use the Django API to find an object matching the URL criteria.
    a = get_object_or_404(articles, pub_date__year=year, pub_date__month=month, pk=article_id)
    return render_to_response('news/article_detail', {'article': a})
```

```
~ -> django-admin.py runserver 8080 —settings=myproject.settings
```

Django: URLS and views



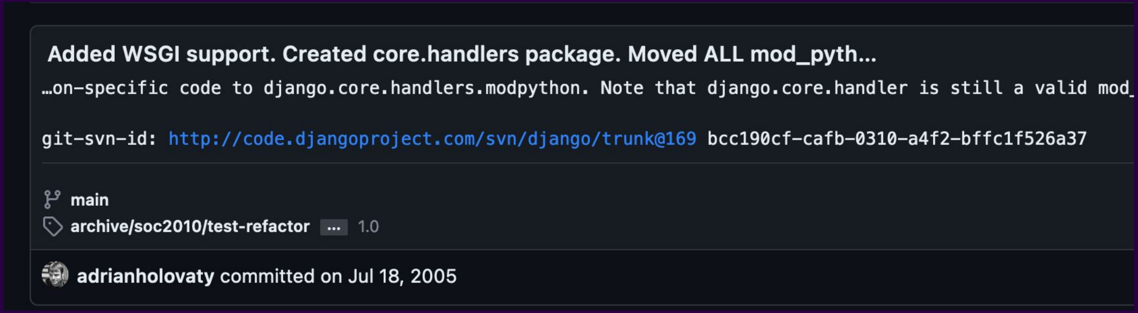
Lars Marius Garshol July 31, 2005 at 3:20 p.m.

What do I do if I want to try Django with a database I already have? Can I say "build me the API from the DB"? Or, alternatively, create the same declarations you show above, and then say "build me the API (assuming the DB is there and that the declarations match)"?

Adrian Holovaty August 1, 2005 at 1:53 p.m.

Lars: That's on the to-do list. See http://code.djangoproject.com/ticket/90 .

Django: The best docs and it had comments!



Added WSGI support. Created core.handlers package. Moved ALL mod_pyth...

```
…on-specific code to django.core.handlers.modpython. Note that django.core.handler is still a valid mod_

git-svn-id: http://code.djangoproject.com/svn/django/trunk@169 bcc190cf-cafb-0310-a4f2-bffc1f526a37
```

main

archive/soc2010/test-refactor ... 1.0

adrianholovaty committed on Jul 18, 2005

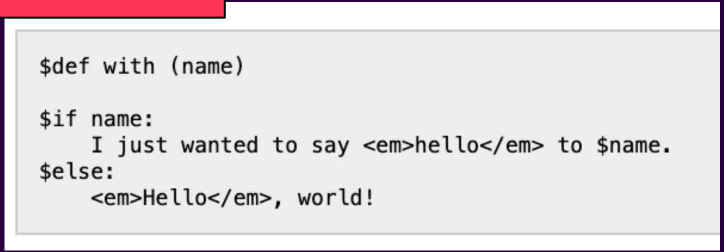
Django added support for WSGI in July 2006. This was pretty quick!

Django: Move to WSGI

ANOTHER ONE

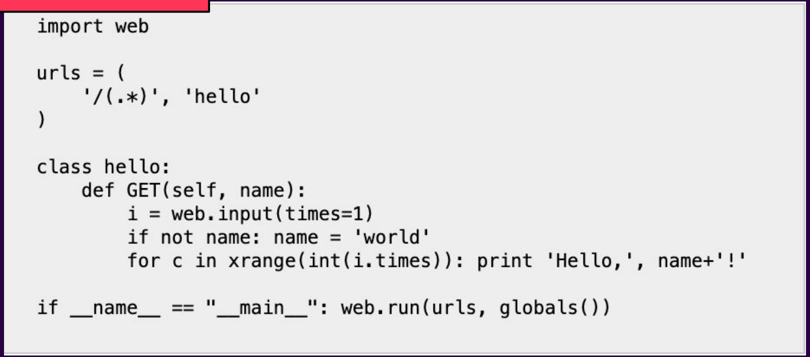
Web.py [2005]

cheetah.html



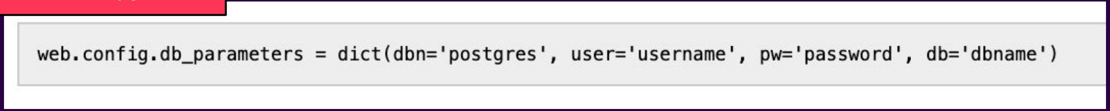
```
$def with (name)

$if name:
    I just wanted to say <em>hello</em> to $name.
$else:
    <em>Hello</em>, world!
```

simple_app.py



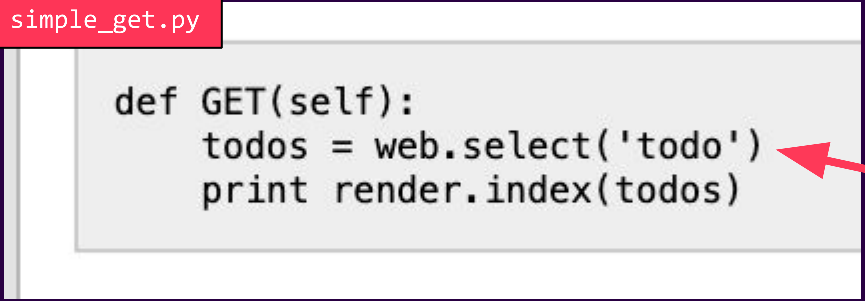
```
import web

urls = (
    '/(.*)', 'hello'
)

class hello:
    def GET(self, name):
        i = web.input(times=1)
        if not name: name = 'world'
        for c in xrange(int(i.times)): print 'Hello,', name+'!'

if __name__ == "__main__": web.run(urls, globals())
```

db.py



```
web.config.db_parameters = dict(dbn='postgres', user='username', pw='password', db='dbname')
```

simple_get.py



```
def GET(self):
    todos = web.select('todo')
    print render.index(todos)
```

You would need to create the todo table yourself.

Web.py: A Simple Framework

Web.py was one of the **earliest adopters** of WSGI. But it launched with FastCGI and Lighttpd. Many frameworks in general used flup to serve WSGI over FastCGI and SCGI.

Web.py: A Simple Framework (that YouTube used)

ANOTHER ONE

Pylons [2005]

Pylons: A Legacy of Modular Design

- Pylons never gained widespread traction
- Encouraged and emphasises modular design
  - Any WSGI compatible server
  - Any templating engine
  - Any ORM
  - Any WSGI middleware
  - You could even use a different router
- It was one of the biggest proponents of the WSGI standard
- It gave birth to Pyramid, which is actively developed today

mod_wsgi [2007]

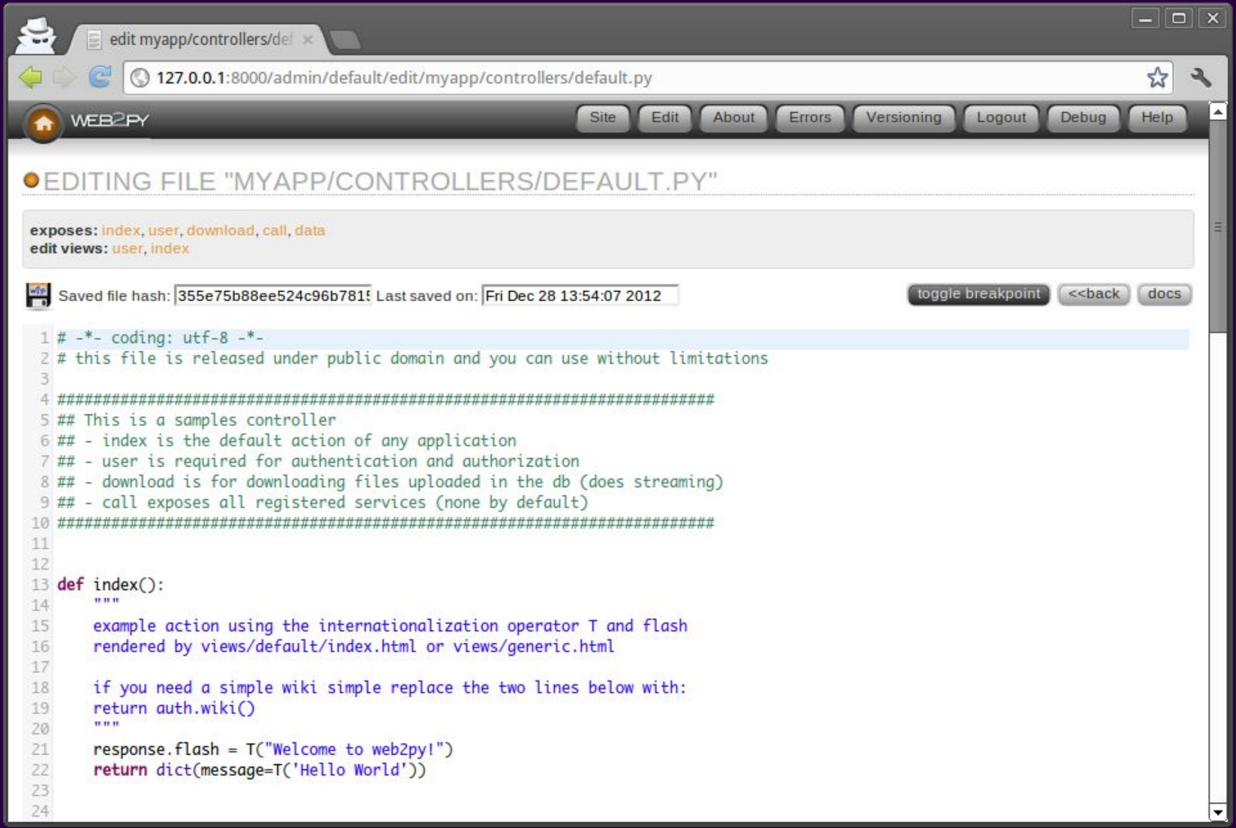
mod_wsgi: Accelerating WSGI adoption

uWSGI [2008]

uWSGI: A Powerful WSGI Reverse Proxy

- Nginx had native support for uWSGI
- Created a powerful WSGI server for all Python WSGI frameworks
- Had Emperor Mode!
- High Performance!
- A lot more …

Web2Py [2007]


edit myapp/controllers/def
127.0.0.1:8000/admin/default/edit/myapp/controllers/default.py
WEB2PY
Site
Edit
About
Errors
Versioning
Logout
Debug
Help
toggle breakpoint
<<back
docs

EDITING FILE "MYAPP/CONTROLLERS/DEFAULT.PY"

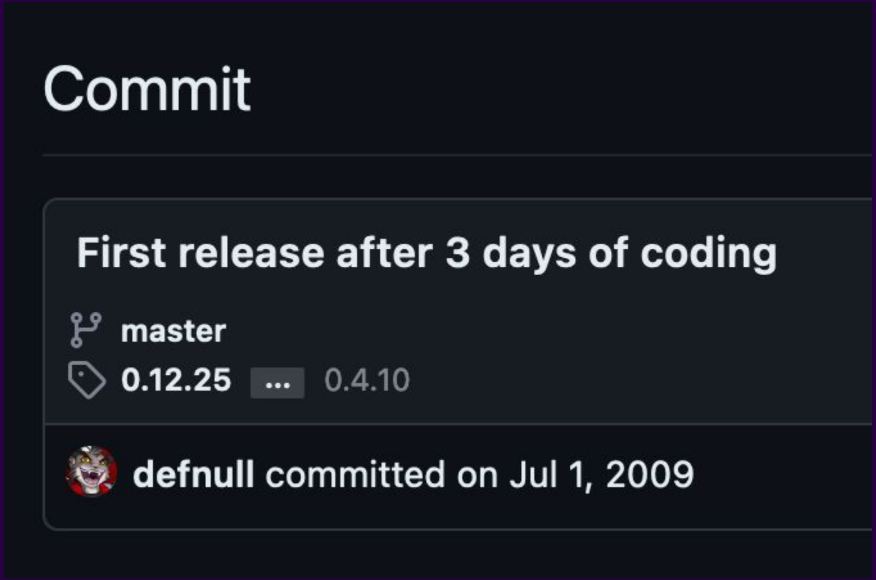
exposes: index, user, download, call, data
edit views: user, index

Saved file hash: 355e75b88ee524c96b781! Last saved on: Fri Dec 28 13:54:07 2012

```
# -*- coding: utf-8 -*-
# this file is released under public domain and you can use without limitations

#########################################################################
## This is a samples controller
## - index is the default action of any application
## - user is required for authentication and authorization
## - download is for downloading files uploaded in the db (does streaming)
## - call exposes all registered services (none by default)
#########################################################################


def index():
    """
    example action using the internationalization operator T and flash
    rendered by views/default/index.html or views/generic.html

    if you need a simple wiki simple replace the two lines below with:
    return auth.wiki()
    """
    response.flash = T("Welcome to web2py!")
    return dict(message=T('Hello World'))


```

Web2Py: Online editor!

Bottle [2009]


Commit

First release after 3 days of coding

master

0.12.25 ... 0.4.10

defnull committed on Jul 1, 2009

Bottle: A framework in 600 lines of code

```python
from bottle import route, run

@route('/hello/:name')
def hello(name):
    return '<h1>Hello %s!</h1>' % name.title()

run(host='localhost', port=8080)
```

Bottle: Everything in a single file!

Tornado [2009]

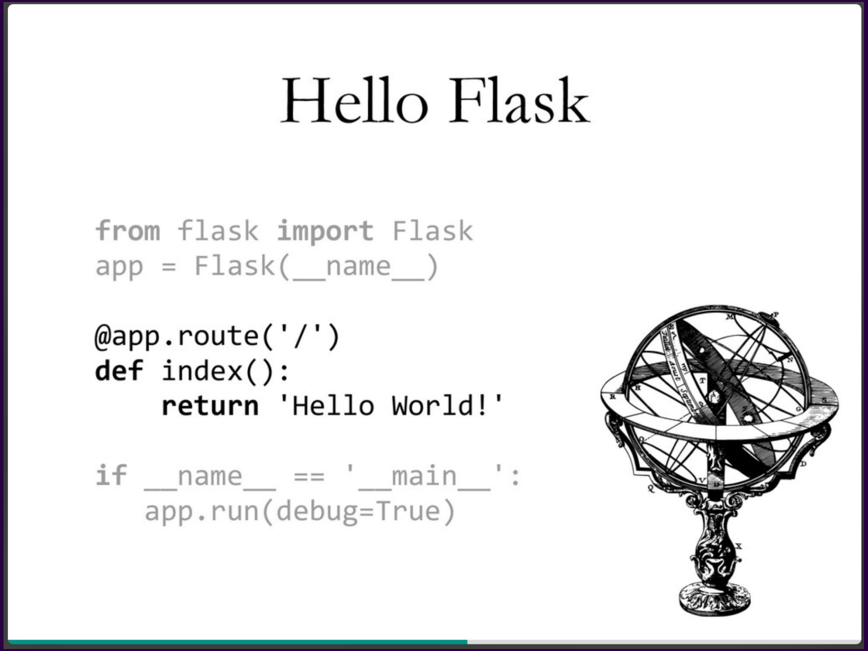
NGINX

```python
import tornado.httpserver
import tornado.ioloop
import tornado.web

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.write("Hello, world")

application = tornado.web.Application([
    (r"/", MainHandler),
])

if __name__ == "__main__":
    http_server = tornado.httpserver.HTTPServer(application)
    http_server.listen(8888)
    tornado.ioloop.IOLoop.instance().start()
```

Handled long-lived connections well!

Tornado: A Renaissance in Async Python Web Frameworks!

Flask [2010]



Hello Flask

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

@app.route('/')
def index():
    return 'Hello World!'

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

Fits in Twitter's character limit!

Flask: The Microframework of Champions!

Why did Flask win over bottle?

- Good idea.
- Start micro, end macro.
- Great documentation
- Extensibility
- Support and Maintenance
- Great development server
- Good error handling capabilities
- All in 450 lines of code.

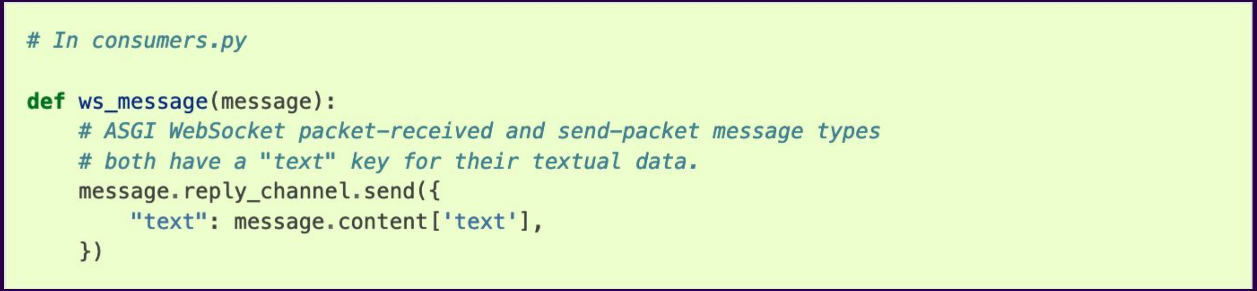
Gunicorn [2010]

Gunicorn

- Fast
- Easy to use
- Used greenlets: Many Connections!
- Pure Python

The Rise of Node.js and WebSockets!

ASGI [2015/2016]



```python
# In consumers.py

def ws_message(message):
    # ASGI WebSocket packet-received and send-packet message types
    # both have a "text" key for their textual data.
    message.reply_channel.send({
        "text": message.content['text'],
    })
```

ASGI: Humble Beginnings from Django Channels

```python
def wsgi_app(environ, start_response):
    status = '200 OK'
    headers = [('Content-type', 'text/plain')]
    start_response(status, headers)
    return [b"Hello, WSGI World!"]
```

```python
async def asgi_app(scope, receive, send):
    await send({
        'type': 'http.response.start',
        'status': 200,
        'headers': [
            (b'Content-type', b'text/plain')
        ]
    })
    await send({
        'type': 'http.response.body',
        'body': b"Hello, ASGI World!"
    })
```

WSGI to ASGI

```
{
    'type': 'http',
    'http_version': '1.1',
    'method': 'GET',
    'path': '/some/path/',
    'root_path': '',
    'scheme': 'http',
    'query_string': b'param1=value1&param2=value2',
    'headers': [
        (b'host', b'www.example.com'),
        (b'user-agent', b'curl/7.64.0'),
        (b'accept', b'*/*'),
    ],
    'client': ('127.0.0.1', 12345),
    'server': ('127.0.0.1', 80),
    'asgi': {
        'version': '3.0',
        'spec_version': '2.1',
    }
}
```

```
async def asgi_app(scope, receive, send):
    await send({
        'type': 'http.response.start',
        'status': 200,
        'headers': [
            (b'Content-type', b'text/plain')
        ]
    })
    await send({
        'type': 'http.response.body',
        'body': b"Hello, ASGI World!"
    })
```

ASGI: Scope [HTTP 1.1]

```
{
        'type': 'websocket',
        'asgi': {
            'version': '3.0',
            'spec_version': '2.1',
        },
        'http_version': '1.1',
        'path': '/ws/somepath/',
        'root_path': '',
        'scheme': 'ws',
        'query_string': b'param1=value1&param2=value2',
        'headers': [
            (b'host', b'www.example.com'),
            (b'sec-websocket-key', b'dGhlIHNhbXBsZSBub25jZQ=='),
            (b'sec-websocket-version', b'13'),
        ],
        'client': ('127.0.0.1', 12345),
        'server': ('127.0.0.1', 8000),
        'subprotocols': [],
        'extensions': {
            'permessage-deflate': {}
        }
}
```

```python
async def asgi_app(scope, receive, send):
    await send({
        'type': 'http.response.start',
        'status': 200,
        'headers': [
            (b'Content-type', b'text/plain')
        ]
    })
    await send({
        'type': 'http.response.body',
        'body': b"Hello, ASGI World!"
    })
```

ASGI: Scope [Websocket]

Starlette [2018]

```python
from starlette.applications import Starlette
from starlette.responses import PlainTextResponse
import uvicorn

app = Starlette()

@app.route('/')
async def hello(request):
    return PlainTextResponse('Hello, World!')

if __name__ == '__main__':
    uvicorn.run(app, host='0.0.0.0', port=8000)
```

Starlette: The ASGI Toolkit

fastapi/applications.py

```python
from fastapi.openapi.utils import get_openapi
from fastapi.params import Depends
from fastapi.types import DecoratedCallable, IncEx
from fastapi.utils import generate_unique_id
from starlette.applications import Starlette
from starlette.datastructures import State
from starlette.exceptions import HTTPException
from starlette.middleware import Middleware
from starlette.middleware.base import BaseHTTPMiddleware
from starlette.middleware.errors import ServerErrorMiddleware
from starlette.middleware.exceptions import ExceptionMiddleware
from starlette.requests import Request
from starlette.responses import HTMLResponse, JSONResponse, Response
from starlette.routing import BaseRoute
from starlette.types import ASGIApp, Lifespan, Receive, Scope, Send

AppType = TypeVar("AppType", bound="FastAPI")
```

Starlette: How it is used in FastAPI

FastAPI [2018]

```python
from fastapi import FastAPI

app = FastAPI()

fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"},
{"item_name": "Baz"}]


@app.get("/items/")
async def read_item(skip: int = 0, limit: int = 10):
    return fake_items_db[skip : skip + limit]
```

```python
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()


class Item(BaseModel):
    name: str
    description: str | None = None
    price: float
    tax: float | None = None

@app.put("/items/{item_id}")
async def update_item(item_id: int, item: Item):
    results = {"item_id": item_id, "item": item}
    return results
```

FastAPI: Integration with Typehints and PyDantic

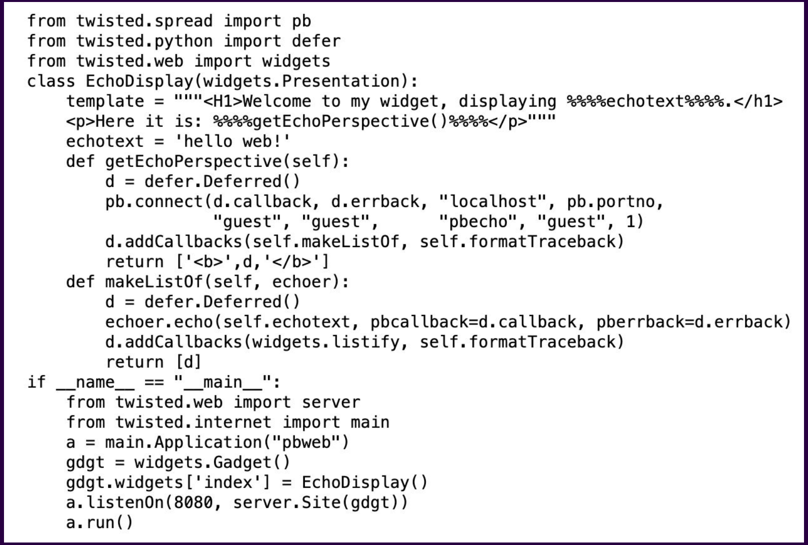
We stand on the shoulders of giants. We have inherited a legacy of ordinary people building extraordinary things. Will we live up to that inheritance?

Questions?

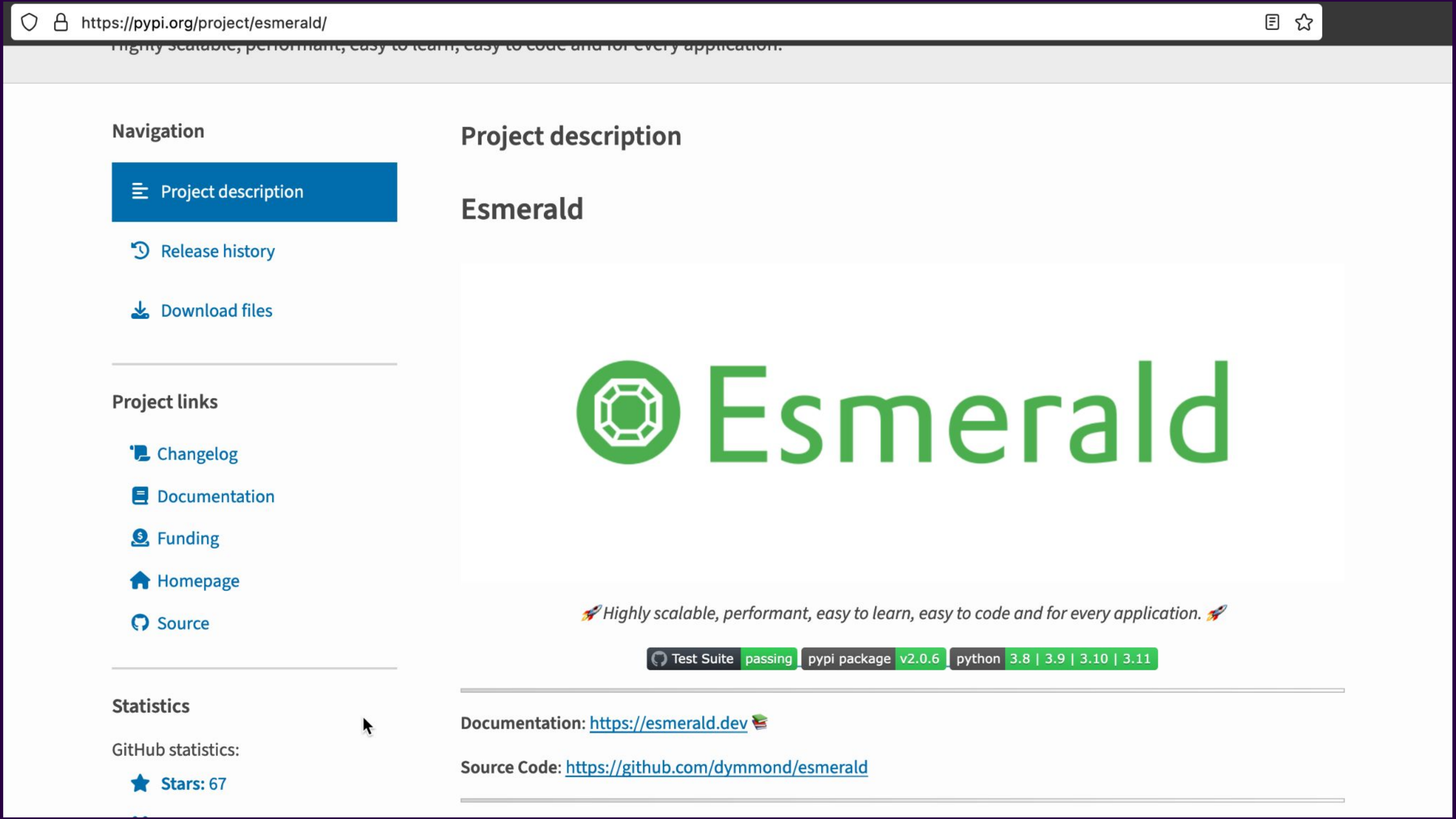
TwistedMatrix [2002]

The first Asynchronous Python Everything

- Twisted is a framework for writing asynchronous, event-driven networked programs in Python.



```
from twisted.spread import pb
from twisted.python import defer
from twisted.web import widgets
class EchoDisplay(widgets.Presentation):
    template = """<H1>Welcome to my widget, displaying %%%%echotext%%%%.</h1>
    <p>Here it is: %%%%getEchoPerspective()%%%%</p>"""
    echotext = 'hello web!'
    def getEchoPerspective(self):
        d = defer.Deferred()
        pb.connect(d.callback, d.errback, "localhost", pb.portno,
                   "guest", "guest",      "pbecho", "guest", 1)
        d.addCallbacks(self.makeListOf, self.formatTraceback)
        return ['<b>',d,'</b>']
    def makeListOf(self, echoer):
        d = defer.Deferred()
        echoer.echo(self.echotext, pbcallback=d.callback, pberrback=d.errback)
        d.addCallbacks(widgets.listify, self.formatTraceback)
        return [d]
if __name__ == "__main__":
    from twisted.web import server
    from twisted.internet import main
    a = main.Application("pbweb")
    gdgt = widgets.Gadget()
    gdgt.widgets['index'] = EchoDisplay()
    a.listenOn(8080, server.Site(gdgt))
    a.run()
```


https://pypi.org/project/esmerald/

Highly scalable, performant, easy to learn, easy to code and for every application.

Navigation

- Project description
- Release history
- Download files

Project links

- Changelog
- Documentation
- Funding
- Homepage
- Source

Statistics

GitHub statistics:

- **Stars:** 67

Project description

Esmerald

Esmerald

🚀 *Highly scalable, performant, easy to learn, easy to code and for every application.* 🚀

Test Suite passing | pypi package v2.0.6 | python 3.8 | 3.9 | 3.10 | 3.11

Documentation: https://esmerald.dev 📚

Source Code: https://github.com/dymmond/esmerald
