Unit Testing Your Web Application

Moshe Zadka – https://cobordism.com

Acknowledgement of Country

Belmont (in San Francisco Bay Area Peninsula)
Ancestral homeland of the Ramaytush Ohlone people

Outline

Outline

Intro to Pyramid
What are Unit tests?

Outline

Intro to Pyramid
What are Unit tests?
What is httpx?

Outline

Intro to Pyramid
What are Unit tests?
What is httpx?
Unit test examples!

Outline

Intro to Pyramid
What are Unit tests?
What is httpx?
Unit test examples!

Crash Course in Pyramid

Some quick examples!

JSON app: View Function

```
def jsonv(request):
    return {}
```

JSON app: Configurator

```
with pyramid.config.Configurator() as config:
    config.add_route("json", "/json")
    config.add_view(jsonv, route_name="json",
                    renderer="json")
```

JSON app: WSGI app

```
app = config.make_wsgi_app()
```

JSON View Retrieval

```
request = client.get("/json")
request.json()

{}
```

Matcher View

```
def matches(request):
    return dict(name=request.matchdict["name"])
with pyramid.config.Configurator() as config:
    config.add_route("matches", "/matches/{name}")
    config.add_view(matches, route_name="matches",
                    renderer="json")
app = config.make_wsgi_app()
```

Matcher View Retrieval

```
request = client.get("/matches/hello")
request.json()

{'name': 'hello'}
```

Settings View

```
def setting(request):
    return dict(
        field=request.registry.settings["field"]
    )
with pyramid.config.Configurator(
    settings=dict(field="field_value")
) as config:
    config.add_route("setting", "/setting")
    config.add_view(setting, route_name="setting",
                    renderer="json")
app = config.make_wsgi_app()
```

Settings View Retrieval

```
request = client.get("/setting")
request.json()

{'field': 'field_value'}
```

View Function

```
def lookup(
    request: pyramid.request.Request
) -> str:
#    ^^^
#    Will be jsonified
    matcher = request.registry.settings["matcher"]
#             ^^^^^^^^^^^^^^^^^^^^^^^^^^
#             Access settings through request
    name = request.matchdict["name"]
#          ^^^^^^^^^^^^^^^^^^
#          Get parameter from route
    return matcher.get(name.lower(), "default")
#          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
#          Business logic
```

Routing

```
def include_lookup(config):
    config.add_route("lookup", "/lookup/{name}")
#                    ^^^^^^^^|^^^^^^^^    ^^^^^^^^^^^^^^^^^^
#                    add a new|route      route path
#                    route    |name
    config.add_view(lookup, route_name="lookup",
#                   ^^^^^^^^|^^^^^    ^^^^^^^^^^^^^^^^^^^^^^^
#                   add a    |view     route to attach
#                   view to  |callable
#                   a route  |
                             renderer="json")
#                            ^^^^^^^^^^^^^^^^
#                            Convert to a JSON response
```

Settings

```
settings = dict(
#^^
#settings
      matcher=dict(special="hello"),
#     ^^^^^^^ ^^^^^^^^^^^^^^^^^^^^^^
#     settings ^^^^^^^^^^^^^^^^^^^^^^
#     key          Mapping "special"
#                  name to "hello"
)
```

Configuration

```
with pyramid.config.Configurator(
#^^^ ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
#Commit at the end        ^^^^^^^^^^^^
#                         Configurator
#                         class
    settings=settings,
#   ^^^^^^^^  ^^^^^^^^^
#   settings  pre-configured
#             dict
) as config:
#    ^^^^^^
#    configurator
#    object
    include_lookup(config)
```

App creation

```
app = config.make_wsgi_app(
#                          ^^^^
#              Web Standard Gateway Interface
)
```

Unit test: rough definition

Unit test: rough definition

Runs the code

Unit test: rough definition

Runs the code
Might fail on buggy code

Unit test: rough definition

Runs the code
Might fail on buggy code
Self-contained

Mock

Mock

Fake object

Mock

Fake object
Configurable behavior

Mock

Fake object
Configurable behavior
Records access

Good Unit Test

Good Unit Test

Avoid failing on valid:

Good Unit Test

Avoid failing on valid:
Public APIs

Good Unit Test

Avoid failing on valid:
Public APIs
Reduce assumptions

Patch Makes Bad Unit Test

Patch Makes Bad Unit Test

Patch: Temporarily replacing a global

Patch Makes Bad Unit Test

Patch: Temporarily replacing a global
Assumption: Global used

Patch Makes Bad Unit Test

Patch: Temporarily replacing a global
Assumption: Global used
Assumption: Used through path

Unit test anatomy

Unit test anatomy

Set-up

Unit test anatomy

Set-up
Execution

Unit test anatomy

Set-up
Execution
Post-process

Unit test anatomy

Set-up
Execution
Post-process (optional)

Unit test anatomy

Set-up
Execution
Post-process (optional)
Verification

WSGI

Framework

WSGI

Framework
to server

httpx

Requests alternative

httpx

Requests alternative
Can consume WSGI directly

httpx with WSGI

Creating the client:

httpx with WSGI

Creating the client:

```
client = httpx.Client(
    base_url="https://example.com/",
    app=app,
)
```

httpx with WSGI

Calling the client:

httpx with WSGI

Calling the client:

```
response = client.get("/lookup/SPECIAL")
```

httpx with WSGI

Using the httpx response object:

httpx with WSGI

Using the httpx response object:

```
response.json()
```

```
'hello'
```

System Under Test: View

```
def lookup(
    request: pyramid.request.Request
) -> str:
    matcher = request.registry.settings["matcher"]
    name = request.matchdict["name"]
    return matcher.get(name, "default")
#                            ^^^^^
#                            Bug: missing .lower()
```

System Under Test: App

```
def make_app(special_value):
#^^                ^^^^^^^^^^^^^^
#^^                Parameter for
#                  the application
#Creating the app from a function,
#not at top level
    settings = dict(
        matcher=dict(special=special_value),
    )
    with pyramid.config.Configurator(
        settings=settings
    ) as config:
        include_lookup(config)
    return config.make_wsgi_app()
```

Calling function directly

Setup:

```
request = mock.MagicMock()
request.registry.settings = dict(
    matcher=dict(special="hello"),
)
request.matchdict = dict(name="SPECIAL")
```

Calling function directly

Execute:

```
result = lookup(request)
```

Calling function directly

Verification

```
try:
    assert_that(result, equal_to("hello"))
except AssertionError as exc:
    print(exc)
```

```
Expected: 'hello'
     but: was 'default'
```

Calling WSGI

Setup:

```
environ = dict(
    PATH_INFO="/lookup/SPECIAL",
    REQUEST_METHOD="GET",
)
start_response = mock.MagicMock(
    return_value=io.BytesIO()
)
```

Calling WSGI

Execute:

```
app = make_app("hello")
base_parts = app(environ, start_response)
```

Calling WSGI

Post-process:

```
parts = [start_response.return_value.getvalue()]
parts.extend(base_parts)
args, kwargs = start_response.call_args
status, headers = args
result = json.loads(b"".join(parts).decode("utf-8")
```

Calling WSGI

Verification

```
try:
    assert_that(result, equal_to("hello"))
except AssertionError as exc:
    print(exc)
```

```
Expected: 'hello'
     but: was 'default'
```

Using httpx

Setup:

```
app = make_app("hello")
client = httpx.Client(
    base_url="https://example.com/",
    app=app,
)
```

Using httpx

Execute:

```
resp = client.get("/lookup/SPECIAL")
```

Using httpx

Post-process:

```
result = resp.json()
```

Using httpx

Verification:

```
try:
    assert_that(result, equal_to("hello"))
except AssertionError as exc:
    print(exc)

Expected: 'hello'
     but: was 'default'
```

Summary: Pyramid

Summary: Pyramid

View:

Summary: Pyramid

View: request, response, route

Summary: Pyramid

View: request, response, route
Route:

Summary: Pyramid

View: request, response, route
Route: URL match

Summary: Pyramid

View: request, response, route
Route: URL match
Registry:

Summary: Pyramid

View: request, response, route
Route: URL match
Registry: Parameters, shared objects

Summary: Pyramid

View: request, response, route
Route: URL match
Registry: Parameters, shared objects
Configurator:

Summary: Pyramid

View: request, response, route
Route: URL match
Registry: Parameters, shared objects
Configurator: Views, Routes, Registry

Summary: Pyramid

View: request, response, route
Route: URL match
Registry: Parameters, shared objects
Configurator: Views, Routes, Registry
WSGI:

Summary: Pyramid

View: request, response, route
Route: URL match
Registry: Parameters, shared objects
Configurator: Views, Routes, Registry
WSGI: Configurator to server

Summary: Pyramid

View: request, response, route
Route: URL match
Registry: Parameters, shared objects
Configurator: Views, Routes, Registry
WSGI: Configurator to server (or httpx!)

Summary: Unit Test

Summary: Unit Test

Verify code,

Summary: Unit Test

Verify code, quickly,

Summary: Unit Test

Verify code, quickly, and safely:

Summary: Unit Test

Verify code, quickly, and safely:
Mock!

Summary: Unit Test

Verify code, quickly, and safely:
Mock!
Pre-plan

Summary: Unit Test

Verify code, quickly, and safely:
Mock!
Pre-plan
Prefer stable APIs

Bonus

Extra! Extra!

View

Function

View

Function
Argument: Request

View

Function
Argument: Request
Return value: Response

View Route

Route

View Route

Route
to View

View Route

Route
to View

View Route Predicates

Route to view can be conditional:

View Route Predicates

Route to view can be conditional:
Request type,

View Route Predicates

Route to view can be conditional:
Request type,
Content type,

View Route Predicates

Route to view can be conditional:
Request type,
Content type,
Arbitrary predicates

Route

Path

Route

Path
to route name

Route

Path
to route name

Route: advanced

Route: advanced

Path fragment matching

Route: advanced

Path fragment matching
...and more

Registry: Application Parameters

Registry: Application Parameters

Settings: ad-hoc dictionary

Configurator

Routes

Configurator

Routes
Views

Configurator

Routes
Views
Registry

Configurator

Routes
Views
Registry

Static Value

```
def empty(request):
    return pyramid.response.Response(
        json.dumps({}).encode("ascii"),
        content_type="application/json",
    )
with pyramid.config.Configurator() as config:
    config.add_route("root", "/")
    config.add_view(empty, route_name="root")
app = config.make_wsgi_app()
```

Static View Retrieval

```
request = client.get("/")
request.json()

{}
```

Parameter View

```
def params(request):
    return dict(thing=request.params["thing"])
with pyramid.config.Configurator() as config:
    config.add_route("params", "/params")
    config.add_view(params, route_name="params", re
app = config.make_wsgi_app()
```

Parameter View Retrieval

```
request = client.get("/params?thing=hello")
request.json()

{'thing': 'hello'}
```

Body View

```
def body(request):
    return dict(field=request.json_body["field"])
with pyramid.config.Configurator() as config:
    config.add_route("body", "/body")
    config.add_view(body, route_name="body", render
app = config.make_wsgi_app()
```

Body View Retrieval

```
request = client.post("/body", json=dict(field="hel
request.json()
```

```
{'field': 'hello'}
```

Registry: Application Parameters

Registry: Application Parameters

Utility: Scalable configuration

Interfaces and Components

```
class IValueGetter(zope.interface.Interface):
    def get_value(self) -> int:
        ...

@zope.interface.implementer(IValueGetter)
@attrs.frozen
class ValueStorer:
    _value: int

    def get_value(self):
        return self._value
```

Interface-based Registry

```
config = pyramid.config.Configurator()
config.registry.registerUtility(ValueStorer(value=4
config.registry.getUtility(IValueGetter).get_value(

42
```

Why Interface-based Registry?

Namespaced

Why Interface-based Registry?

Namespaced
Semantically meaningful

Why Interface-based Registry?

Namespaced
Semantically meaningful
Configuration

Configurator Inclusion

```
def root_stuff(config):
    config.add_route("root", "/")
    config.add_view(empty, route_name="root")
with pyramid.config.Configurator() as config:
    config.include(root_stuff)
    app = config.make_wsgi_app()

request = client.get("/")
request.json()

{}
```

Configurator Inclusion

Include allows:

Configurator Inclusion

Include allows:

- Callable

Configurator Inclusion

Include allows:

- Callable
- Callable dotted name

Configurator Inclusion

Include allows:

- Callable
- Callable dotted name
- Module (will use "includeme")

Configurator Inclusion

Include allows:

- Callable
- Callable dotted name
- Module (will use "includeme")
- Module dotted name

Configurator Scanning

* Scan a package * Find functions decorated "view config"