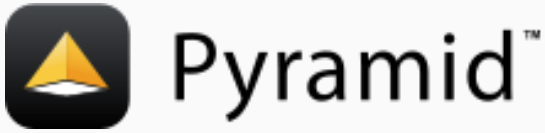


Pyramid

Framework python pour le développement d'applications Web
Présentation et retour d'expérience

Philippe Bollard
philippe.bollard@univ-grenoble-alpes.fr

7 juin 2016

CNRS/IPAG

Plan

1. Présentation de Pyramid
2. Quickstart
3. Démarrer un nouveau projet
4. Mapping URL
5. Modèle et ORM
6. Vues
7. Moteur de templates
8. Autour de Pyramid

Introduction

Contexte projet

SSHADE

- (Sous-)Projet européen (Europlanet H2020, VESPA)
- Infrastructure VO de BDD dédiées à la spectroscopie des solides
- Modèle de données assez complexe (environ 400 pages)

Technos

- PostgreSQL
- Nginx + uWSGI

Liens

- `https://www.sshade.eu` (à partir de cet été)
- `https://blog.sshade.eu`
- `https://forge.sshade.eu`

Présentation de Pyramid

À propos de Pyramid

Liens

- `https://trypyramid.com/`
- `http://www.pylonsproject.org/`
- `https://github.com/Pylons/pyramid`

Pyramid, c'est quoi ?

Framework Python pour applications Web

- parmi les 3 plus connus avec Django et Flask
- mature et assez répandu dans la communauté
- Python 2 et Python 3

À la carte

- coeur minimal
- pas d'ORM par défaut
- extensible par de nombreux modules
- plusieurs alternatives pour une même fonctionnalité

Un peu d'histoire

Diverses inspirations

- Zope
- Repoze.bfg
- Pylons

Historique

- 2005-2011 : Pylons
- 06/2008-11/2010 : Repoze.bfg
- 12/2010 : Fusion de Repoze.bfg avec Pylons

Philosophie

Simplicité

- framework non-monolithique, "pay only for what you eat"

Minimalisme

- coeur focalisé sur les tâches essentielles
- mapping URL > code, templates, sécurité, contenu statique

Fiabilité

- "If it ain't tested, it's broke", couverture de code à 100%

Ouverture

- code "open source", licence très permissive

Documentation

Vitesse

Quickstart

quickstart.py, un Hello World! dans un seul fichier

```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response

def hello_world(request):
    return Response('Hello %(name)s!' % request.matchdict)

if __name__ == '__main__':
    config = Configurator()
    config.add_route('hello', '/hello/{name}')
    config.add_view(hello_world, route_name='hello')
    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 6543, app)
    server.serve_forever()
```

Préparation de l'environnement de travail

Dépendances

- `apt-get install python python-dev python-virtualenv`

Création du dossier

- `mkdir ~/demo-pyramid`
- `cd ~/demo-pyramid`

Initialisation d'un environnement virtuel

- `virtualenv .env`
- `source .env/bin/activate`

Installation de Pyramid

- `pip install pyramid`

3, 2, 1... Lancez !

Créer l'application

- `nano quickstart.py`

Voir le résultat

- `python quickstart.py`
- Ouvrir `http://localhost:6543/hello/toto`

Démarrer un nouveau projet

Créer un projet à partir d'un modèle (sans ORM)

Préparer l'environnement de travail

Créer l'application

- `pcreate -s starter demo`

Configurer

- `cd demo`
- `nano setup.py`
- `nano development.ini`

Lancer

- `python setup.py develop`
- `pserve development.ini --reload`
- Ouvrir `http://localhost:6543/`

Créer un projet à partir d'un modèle (avec SQLAlchemy)

Préparer l'environnement de travail

Créer l'application

- `pcreate -s alchemy demo`

Configurer

- `cd demo`
- `nano setup.py`
- `nano development.ini`

Lancer

- `python setup.py develop`
- `initialize_db development.ini`
- `pserve development.ini --reload`
- Ouvrir `http://localhost:6543/`

Structure d'un projet généré avec le modèle alchemy

```
|- setup.py
|- development.ini
|- production.ini
|- demo
     |- __init__.py
     [- routes.py
     |- models
     |      |- __init.py__
     |      |- meta.py
     |      |- mymodel.py
     |- views
     |      |- default.py
     |- templates
     |      |- layout.jinja2
     |      |- mytemplate.jinja2
     |- scripts
     |- static
```

Configuration

Fichier setup.py

- dépendances à installer
- informations sur l'application

Fichiers development.ini et production.ini

- paramètres de l'application (couples clé=valeur)
- connexion à la base de données
- activation du debug et réglage des niveaux de log
- modules à inclure (par Pyramid)

Extrait du fichier development.ini

```
[app:main]
use = egg:demo

pyramid.reload_templates = true
pyramid.debug_authorization = false
pyramid.debug_notfound = false
pyramid.debug_routematch = false
pyramid.default_locale_name = en
pyramid.includes =
    pyramid_debugtoolbar

sqlalchemy.url = sqlite:///%(here)s/demo.sqlite
```

Point d'entrée : demo/__init__.py

```python
from pyramid.config import Configurator

def main(global_config, **settings):
    """ This function returns a Pyramid WSGI application.
    """
    config = Configurator(settings=settings)
    config.include('pyramid_jinja2')
    config.include('.models')
    config.include('.routes')
    config.scan()
    return config.make_wsgi_app()
```

Mapping URL

Deux façons de mapper les URL

URL Dispatch

- ensemble de règles nommées "routes"
- découpage d'une URL selon des patterns
- peut servir à la génération d'URL

foo/{baz}/{bar}	
foo/1/2	{'baz':u'1', 'bar':u'2'}
foo/abc/def	{'baz':u'abc', 'bar':u'def'}

Traversal

- l'URL est un chemin dans une arborescence
- le modèle doit être adapté en conséquence

Définition des routes : demo/routes.py

```
def includeme(config):
    config.add_static_view('static', 'static', cache_max_age=3600)
    config.add_route('home', '/')

    config.add_route('foo', '/foo/{baz}/{bar}')
```

Modèle et ORM

Plusieurs ORM

Pyramid vous laisse le choix !

- Pas d'ORM
- ORM relationnel
- ORM NoSQL

Modules

- pyramid_sqlalchemy
- pyramid_mongo
- pyramid_pewee
- pyramid_orb

Principal ORM : SQLAlchemy

Liens

- http://www.sqlalchemy.org/
- http://docs.sqlalchemy.org/en/latest/
- https://pyramid-sqlalchemy.readthedocs.io/en/latest/

Outil de migration : Alembic

- https://alembic.readthedocs.io/en/latest/
- https://bitbucket.org/zzzeek/alembic

Fonctionnement du modèle

demo/models/__init__.py

- importe toutes les classes du modèle
- initialise la session et place l'objet dans le registre

demo/models/meta.py

- initialise une meta-classe
- réglage de certains paramètres

demo/models/mymodel.py

- définit les tables, colonnes, relations
- définit implicitement les classes des objets métiers

Exemple

```python
from sqlalchemy import Column, ForeignKey, Index, Integer, Text
from sqlalchemy.orm import relationship
from .meta import Base

class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    name = Column(Text, nullable=False, unique=True)

class Page(Base):
    __tablename__ = 'pages'
    id = Column(Integer, primary_key=True)
    name = Column(Text, nullable=False, unique=True)
    data = Column(Text, nullable=False)

    creator_id = Column(ForeignKey('users.id'), nullable=False)
    creator = relationship('User', backref='created_pages')
```

Utilisation de SQLAlchemy

Sélection

```
pages = DBSession.query(Page).all()
page1 = DBSession.query(Page).get(1)
page2 = DBSession.query(Page).filter_by(name='deux').first()
```

Insertion

```
u = DBSession.query(User).get(5)

p = Page()
p.id = 3
p.name = 'trois'
p.data = 'Les trois mousquetaires'
p.creator = u

DBSession.add(p)

DBSession.commit()
```

Vues

Composition d'une vue

Méthode

- englobée ou non dans une classe
- objet 'request' en paramètre d'entrée
- dictionnaire de variables en sortie

Décorateur @view_config

route_name Route applicable

match_param Filtrage sur un paramètre d'URL (via la route)

renderer Template ou moteur de rendu (json)

permission Permission nécessaire pour accéder à cette vue

`http://docs.pylonsproject.org/projects/pyramid/en/latest/narr/viewconfig.html`

Exemple

```
from pyramid.httpexceptions import HTTPFound, HTTPNotFound
from pyramid.view import view_config
from ..models import Page, User

@view_config(route_name='view_page',
             renderer='../templates/view.jinja2')
def view_page(request):
    pagename = request.matchdict['pagename']
    dbs = request.dbsession
    page = dbs.query(Page).filter_by(name=pagename).first()
    if page is None:
        raise HTTPNotFound('No such page')

    return dict(page=page)
```

Moteur de templates

Plusieurs moteurs de templates

Pyramid vous laisse le choix !

- Pas de moteur
- Jinja2
- Chameleon
- Mako

Modules

- pyramid_jinja2
- pyramid_chameleon
- pyramid_mako

Jinja2

```
<title>{% block title %}{% endblock %}</title>
<ul>
{% for user in users %}
    <li><a href="{{ user.url }}">{{ user.username }}</a></li>
{% endfor %}
</ul>
```

- Syntaxe inspirée de Django
- http://jinja.pocoo.org/docs/latest/
- http://docs.pylonsproject.org/projects/pyramid_jinja2/en/latest/

Chameleon

```
<html>
  <body>
    <h1>Hello, ${'world'}!</h1>
    <table>
      <tr tal:repeat="row 'apple', 'banana', 'pineapple'">
        <td tal:repeat="col 'juice', 'muffin', 'pie'">
          ${row.capitalize()} ${col}
        </td>
      </tr>
    </table>
  </body>
</html>
```

- Syntaxe basée sur XML
- `https://chameleon.readthedocs.io/en/latest/`

Mako

```
<%inherit file="base.html"/>
<%
    rows = [[v for v in range(0,10)] for row in range(0,10)]
%>
<table>
    % for row in rows:
        ${makerow(row)}
    % endfor
</table>

<%def name="makerow(row)">
    <tr>
    % for name in row:
        <td>${name}</td>\
    % endfor
    </tr>
</%def>
```

- Syntaxe similaire à Jinja2 mais plus proche de Python
- `http://www.makotemplates.org/`

Autour de Pyramid

Génération et validation de formulaires

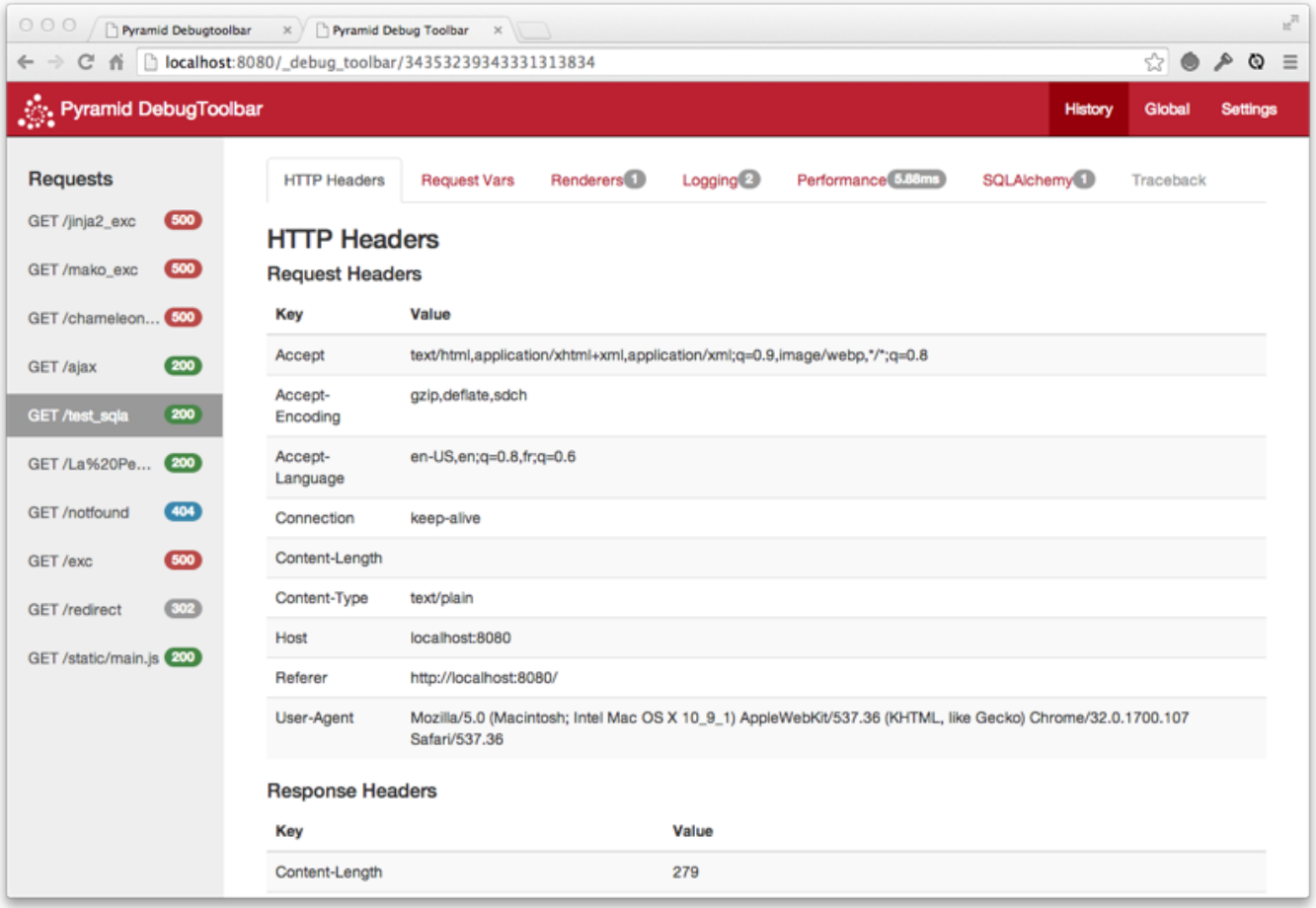
Deform et Colander

- modules les plus courants avec Pyramid
- `http://docs.pylonsproject.org/projects/colander`
- `http://docs.pylonsproject.org/projects/deform`

WTForms

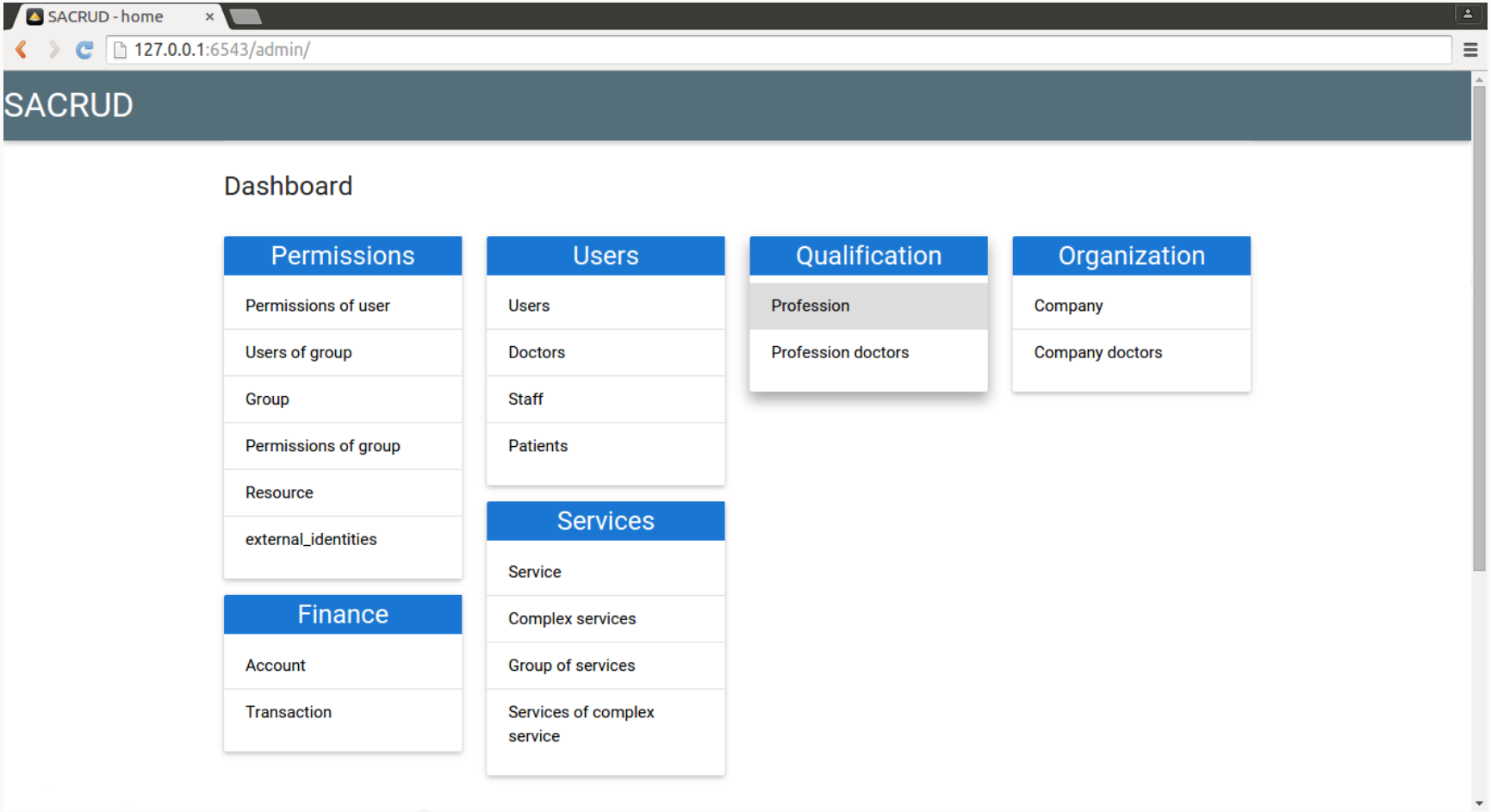
- plus simple à utiliser que colander/deform
- meilleur contrôle du rendu HTML des éléments du formulaire
- `https://wtforms.readthedocs.io/en/latest/`

Interface de debug : pyramid_debugtoolbar



Infos sur : Entêtes HTTP, Exceptions, Performance, Requêtes SQL

Interface d'administration : pyramid-sacrud



http://pyramid-sacrud.readthedocs.io/

Pour aller plus loin...

Autres modules utiles

- http://docs.pylonsproject.org/projects/pyramid_layout/en/latest/
- https://sqlalchemy-utils.readthedocs.io/en/latest/
- https://pint.readthedocs.io/
- http://lxml.de/
- http://pyramid-cubicweb.readthedocs.io/en/latest/

Quelques liens

- http://awesome-pyramid.readthedocs.io/en/latest/
- https://github.com/uralbash/awesome-pyramid
- https://github.com/vinta/awesome-python/

Bilan

Retour d'expérience

Positif

- coeur simple et facile à prendre en main
- permet de commencer petit et de grossir avec le projet
- facile à déployer
- de nombreux modules proposant plusieurs alternatives

Négatif

- documentation de qualité hétérogène et un peu touffue
- beaucoup de temps passé à sélectionner les bons modules (au début)
- attention à la compatibilité Python 3 pour certains modules

Questions ?