The Pyramid Web Application Development Framework

www.pylonsproject.org

Martin Geisler

Dealini

July 4th, 2013

Outline

Outline

Overview

Pyramid is a framework for writing web applications

- inspired by Zope, Pylons, and Django
- scales from very small to very large applications

Overview

Pyramid is a framework for writing web applications

- inspired by Zope, Pylons, and Django
- scales from very small to very large applications
- mature and active open source project
  - very well documented
  - extremely well tested
  - was repoze.bfg until 2010
  - latest version is 1.4.2 from May 2013

Features

Large framework with many features:

- flexible URL dispatching
- supports many template languages
- offers session management
- authentication and authorization
- handles file uploads
- integrated debugging
- ...

Deployment

Pyramid uses WSGI (PEP 333):

- standard way for Python web apps to talk with web servers
- lets you decouple choice of web server from web framework
- implemented for Apache, Nginx, ...

We run Pyramid on `mod_wsgi` under Apache on Amazon EC2.

Persistence

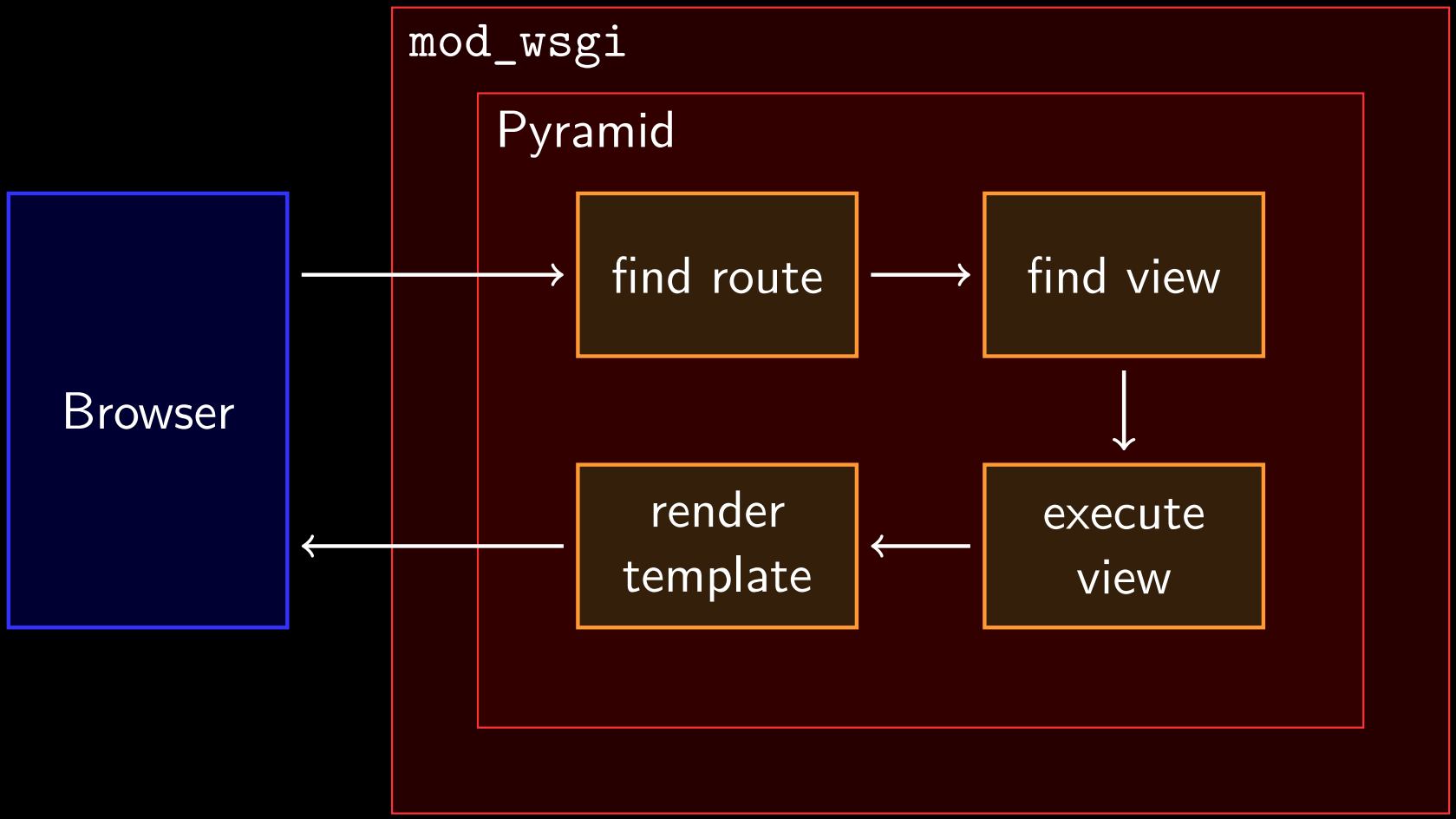
Make your pick:

- no database
- SQLAlchemy
- ...

We use SQLAlchemy with MySQL.

Outline

Request Processing



Routes

Routes define the URLs used by your app:

```
config = Configurator()
config.add_route("home", "/")
config.add_route("post", "/{year}/{month}/{day}/{slug}/")
config.add_route("by_date", "/{year}/{month}/{day}/")
config.add_route("by_month", "/{year}/{month}/")
config.add_route("by_year", "/{year}/")
```

Routes

Routes define the URLs used by your app:

```
config = Configurator()
config.add_route("home", "/")
config.add_route("post", "/{year}/{month}/{day}/{slug}/")
config.add_route("by_date", "/{year}/{month}/{day}/")
config.add_route("by_month", "/{year}/{month}/")
config.add_route("by_year", "/{year}/")
```

Use routes to generate URLs up-to-date in your templates:

```
<a href="${request.route_path("by_year", year=today.year)}">
  More posts from this year
</a>
```

Views

Views produce a Reponse based on a Request:

```
@view_config(route_name="post", renderer="post.mak")
def post_view(request):
    year = int(request.matchdict["year"])
    month = int(request.matchdict["month"])
    day = int(request.matchdict["day"])
    slug = request.matchdict["slug"]
    post = lookup_post(year, month, day, slug)
    if post:
        today = datetime.today()
        return {"today": today, "post": post}
    else:
        raise HTTPNotFound
```

Views

Views produce a Reponse based on a Request:

```
@view_config(route_name="post", renderer="post.mak")
def post_view(request):
    year = int(request.matchdict["year"])
    month = int(request.matchdict["month"])
    day = int(request.matchdict["day"])
    slug = request.matchdict["slug"]
    post = lookup_post(year, month, day, slug)
    if post:
        today = datetime.today()
        return {"today": today, "post": post}
    else:
        raise HTTPNotFound
```

- each view is associated with a route
- routes can have multiple views
- views can have predicates

Renderers

Renderers process output from views:

- return Response object directly:

```
@view_config(route_name="robots_txt")
def robots_txt(request):
    with open("robots.txt"):
        body = fp.read()
        return Response(content_type="text/plain", body=body)
```

Renderers

Renderers process output from views:

- return Response object directly:

```
@view_config(route_name="robots_txt")
def robots_txt(request):
    with open("robots.txt"):
        body = fp.read()
        return Response(content_type="text/plain", body=body)
```

- pass data to a template renderer:

```
@view_config(route_name="home", renderer="home.mak")
def home(request):
    return {"posts": recent_posts()}
```

Renderers

Renderers process output from views:

- return Response object directly:

```
@view_config(route_name="robots_txt")
def robots_txt(request):
    with open("robots.txt"):
        body = fp.read()
        return Response(content_type="text/plain", body=body)
```

- pass data to a template renderer:

```
@view_config(route_name="home", renderer="home.mak")
def home(request):
    return {"posts": recent_posts()}
```

- return JSON data:

```
@view_config(route_name="comments", renderer="json")
def comments(request):
    return {"comments": load_comments(request.params["post"])}
```

Templates

Pyramid has a pluggable API for templates:

- built-in support for Chameleon ZPT and Mako templates
- third-party bindings for Jinja2, Genshi, ...
- template language recognized from renderer kwarg

Outline

Mako Templates

Fast and flexible template language

Mako Templates

Fast and flexible template language

- Switching to Python:

```
<p>Welcome back ${name}!</p>
```

Mako Templates

Fast and flexible template language

- Switching to Python:

```
<p>Welcome back ${name}!</p>
```

- Conditionals:

```
% if requests:
    <p>You have ${len(requests)} friend requests.</p>
% else:
    <p>Nobody wants to be friends with you :-(</p>
% endif
```

Mako Templates

Fast and flexible template language

- Switching to Python:

```
<p>Welcome back ${name}!</p>
```

- Conditionals:

```
% if requests:
    <p>You have ${len(requests)} friend requests.</p>
% else:
    <p>Nobody wants to be friends with you :-(</p>
% endif
```

- Loops:

```
% for request in requests:
    <p class="${loop.cycle('odd', 'even')}>
        Request ${loop.index + 1}: ${request.subject}.
    </p>
% endfor
```

Functions

Mako lets you reuse parts of your templates

- Defining functions:

```
<%def name="format_result(result)">
    <p><a href="${result.href}">${result.title}</a></p>
</%def>
```

Functions

Mako lets you reuse parts of your templates

- Defining functions:

```
<%def name="format_result(result)">
    <p><a href="${result.href}">${result.title}</a></p>
</%def>
```

- This function can now be used elsewhere:

```
% for result in results:
    ${format_result(result)}
% endfor
```

Blocks and Inheritance

Building blocks for page structure:

- Defining a block in a base template:

```
<head>
    <%block name="css">
        <link rel="stylesheet" href="base.css">
    </%block>
</head>
```

Blocks and Inheritance

Building blocks for page structure:

- Defining a block in a base template:

```
<head>
    <%block name="css">
        <link rel="stylesheet" href="base.css">
    </%block>
</head>
```

- Extending a block in an inheriting template:

```
<%inherit file="base.mak"/>

<%block name="css">
    ${parent.css()}
    <link rel="stylesheet" href="extra.css">
</%block>
```

Filtering Output

Mako comes with a number of filters:

- Mako HTML escaping output by default!

```
% if not results:
    <p>Sorry, no results for "${request.GET['q']}".</p>
% endif
```

Filtering Output

Mako comes with a number of filters:

- Mako HTML escaping output by default!

```
% if not results:
    <p>Sorry, no results for "${request.GET['q']}".</p>
% endif
```

- You can disable this with a no-escape filter:

```
<script type="text/javascript">
    show_results(${json.dumps(results) | n})
</script>
```

Outline

Conclusion

Pyramid is fast, flexible and down-to-earth:

- focuses on core features of web design
- has the excellent documentation

Get started easily a scaffold:

```
$ virtualenv pyramid-venv
$ source pyramid-venv/bin/activate
$ pcreate -s starter proj
$ cd proj
$ python setup.py develop
$ pserve development.ini
```

Have fun creating create web sites!

Conclusion

Pyramid is fast, flexible and down-to-earth:

- focuses on core features of web design
- has the excellent documentation

Get started easily a scaffold:

```
$ virtualenv pyramid-venv
$ source pyramid-venv/bin/activate
$ pcreate -s starter proj
$ cd proj
$ python setup.py develop
$ pserve development.ini
```

Have fun creating create web sites!

Thank you! Questions?