Piramid Documentation

Release 1

Wagner

January 28, 2016

Contents

CHAPTER 1

O que é isto?

Este documento visa demonstrar como foi desenvolvida a aplicação aqui denominada PYRAMID. Na verdade, esta é apenas uma aplicação de teste, desenvolvida como treino para o aprendizado do framework Pyramid, utilizando a linguagem de programação PYTHON. Além disso, de maneira simplória, é feita a integração com o SQLAlchemy, que é um framework próprio para interação com base de dados.

CHAPTER 2

Estrutura de arquivos e diretórios

__init__.py – Configuração inicial.

models.py – Camada de acesso á dados.

scripts – Onde se localizam os scripts de atualização ou manutenção da aplicação.

static– Onde ficam os arquivos estáticos.

templates – Arquivos de visualização.

tests.py – Testes automatizados.

views.py – Camada de negócio da aplicação.

CHAPTER 3

Instalação

Tal configuração é para ser feito no terminal de uma máquina linux. A máquina em uso é Centos versão 7. Caso utilize outro tipo de sistema operacional, é necessário rever a síntaxe adotada.

1. yum install sqlite-devel git
2. virtualenv egito
3. cd egito
4. bin/activate ou '. bin/activate'
5. pip install "pyramid==1.6"
6. bin/pcreate -s alchemy tutorial
7. rm -rf tutorial/*
8. git clone https://github.com/wagnerwar/piramid.git tutorial
9. cd tutorial/
10. python setup.py develop
11. cd ../
12. bin/initialize_tutorial_db tutorial/development.ini

CHAPTER 4

Modelo de dados

Arquivo models.py

```
class Video(Base):
    __tablename__ = 'video'
    id = Column(Integer, primary_key=True)
    name = Column(Text)
    descricao = Column(Text)
    preco = Column(Float)

Index('my_index', Video.name, unique=True, mysql_length=255)
```

Arquivo scripts/initializedb.py

```
def main(argv=sys.argv):
    if len(argv) < 2:
        usage(argv)
        config_uri = argv[1]
        options = parse_vars(argv[2:])
        setup_logging(config_uri)
        settings = get_appsettings(config_uri, options=options)
        engine = engine_from_config(settings, 'sqlalchemy.')
        DBSession.configure(bind=engine)
        Base.metadata.create_all(engine)
        with transaction.manager:
            model = Video(name='one', preco=1)
            DBSession.add(model)
```

Se atente para as duas linhas acima, no qual é feita a inserção de um registro para fins de teste.

Sincronização e criação das tabelas bin/initialize_tutorial_db tutorial/development.ini – Onde tutorial/development.ini é o arquivo de configuração

Como visto acima, a classe Video representa a tabela video, com os seguintes atributos: id,name,descricao e preco. O framework SqlAlchemy permite que, não manipulemos dados diretamente, mas, objetos, que representam estes dados. Por isso, não precisa conhecer a estrutura dos dados, apenas, saber usar a camada de abstração destes dados. Os mesmos dados podem estar no SQLite ou no POSTGRESQL; não importa, a forma de manipulá-los é a mesma. Foi criado um índice chamada my_index associado com a tabela video.

CHAPTER 5

Cadastro

Arquivo __init__.py

```
def main(global_config, **settings):
    config.include(videos_include, route_prefix='/videos')

def videos_include(config):
    config.add_route('cadastrar', '/cadastrar')
```

Conforme visto acima, estão configuradas todos os caminhos iniciados com 'videos/' dentro da função videos_include. Por exemplo, quando digitamos 'videos/cadastrar', será carregada a view cadastrar, cujo conteúdo segue abaixo, no arquivo views.py.

Arquivo views.py

```
@view_config(route_name='cadastrar',renderer='templates/cad.pt')
def cadastrar(request):
    save_url = request.route_url('cadastrar')
    request.route_url('consulta')
    if  request.params:
        print('PASSOU')
        nome = request.params['nome']
        descricao = request.params['descricao']
        preco = request.params['preco']
        novo_video = Video(name=nome,descricao=descricao,preco=preco)
        try:
            DBSession.add(novo_video)
            return HTTPFound(location=request.route_url('consulta'))
        except DBAPIError:
        return Response("ERRO DB")
    else:
        print('NAO PASSOU')
        return {'save_url': save_url,'project': 'tutorial'}
```

Conforme visto acima, a view denominada 'cadastrar' renderiza o template 'templates/cad.pt'. A função cadastrar trata das requisições e respostas desta URI (Entende-se como caminho de uma URL, por exemplo: http://localhost/videos/cadastrar) Observe que, se existem parâmetros na requisição, o sistema tenta cadastrar um video novo. Se não há parâmetros, simplesmente exibe o formulário para inclusão de um novo vídeo.

Observe que, em cada view, o 'return' sempre retorna as variáveis que vão para o template. Por exemplo, na função cadastrar() definimos que 'save_url' será equivalente ao valor da variável local save_url (return {'save_url': save_url}). Por isso, tal valor é acessível no template abaixo(<form action="${save_url}" method="GET">)

Arquivo de template: templates/cad.pt (Trecho relevante)

```
<div class="content">
<h1><span class="font-semi-bold">Cadastro de vídeos</span></h1>
<form action="${save_url}" method="GET">
<label>Nome:<br>
<input type="text" name="nome" value="" /><br>
</label>
<label>Descricao:<br>
<input type="text" name="descricao" value="" /><br>
</label>
<label>Preco:<br>
<input type="text" name="preco" value="" /><br>
</label>
<label>
<input type="submit" value="Cadastrar" style="margin-top: 1.2em;">
</label>
</form>
</div>
```

No meio de um grande código HTML, entre tags HTML e BODY, depois da tag HEAD, segue acima o que realmente nos interessa.

URL: http://192.168.56.101:6543/videos/cadastrar

CHAPTER 6

Consulta

Arquivo __init__.py

```
def videos_include(config):
    config.add_route('consulta', '/')
```

Agora, no trecho acima, foi configurada nova rota. Ou seja, quando digitarmos na barra de endereço “videos/”, seremos redirecionados para a view ‘consulta’. A configuração desta view segue abaixo:

Arquivo views.py

```
@view_config(route_name='consulta',renderer='templates/consulta.pt')
    def consulta(request):
        videos = DBSession.query(Video).all()
        url_edit = request.route_url('edicao')
        url_cad = request.route_url('cadastrar')
        return {'videos': videos,'url_edit': url_edit,'url_cad': url_cad}
```

Conforme código acima, eu busco todos os registros da tabela video, para exibi-los numa listagem.

Arquivo de template: ‘templates/consulta.pt’ (Trecho relevante)

```
<div class="content">
<h1>Listagem de vídeos</h1>
<a tal:attributes="href string:${url_cad}"><button>CADASTRAR</button></a>
<div tal:repeat="item videos">
<div class="vido">
<a tal:attributes="href string:${url_edit}?&id=${item.id} "><strong>Nome: </strong><span tal:content=
<strong>Descricao: </strong><span tal:content="string:${item.descricao}" /><br />
<strong>Preco: </strong><span tal:content="string:${item.preco}" /><br />
</div>
</div>
</div>
</div>
```

Acima, a listagem de vídeos.

CHAPTER 7

Edição

Arquivo __init__.py

```
def videos_include(config):
    config.add_route('edicao', '/editar')
```

Segue acima, a configuração da rota 'videos/editar'.

Arquivo views.py

```
@view_config(route_name='edicao',renderer='templates/edicao.pt')
    def editar(request):
        save_url = request.route_url('edicao')
        dell = request.route_url('exclusao')
        id = request.params['id']
        video = DBSession.query(Video).filter_by(id=id).one()
        if 'nome' in request.params.keys():
            try:
                print("PASSOU")
                nome = request.params['nome']
                descricao=request.params['descricao']
                preco=request.params['preco']
                dados = DBSession.query(Video).filter_by(id=id).update({'name': nome,'descricao': des
                return HTTPFound(location=request.route_url('consulta'))
            except Exception:
                return Response('ERRO DB')
            else:
                print("nao passou")
                return {'save_url': save_url,'video': video,'dell': dell}
```

Nesta view, verifica se existem parâmetros que identifiquem que a requisição se refere á submissão de um formulário. Se sim, é feita a atualização do video em questão, identificado pelo atributo 'id'. Se não, é carregado um formulário com os campos para edição do registro.

Arquivo de template: 'templates/edicao.pt' (Trecho relevante)

```
<div class="content">
<h1><span class="font-semi-bold">EDICAO</span> <span class="smaller">Videos</span></h1>
<form action="${save_url}" method="GET">
<label>Nome:<br>
<input type="text" name="nome" value="${video.name}" /><br></label>
<label>Descricao:<br>
<input type="text" name="descricao" value="${video.descricao}" /><br>
</label><label>Preco:<br>
<input type="text" name="preco" value="${video.preco}" /><br>
```

```
</label>
<label>
<input tal:attributes="type string:hidden; name string:id; value string:${video.id}">
<input type="submit" value="Editar" style="margin-top: 1.2em;">
<a tal:attributes="href string:${dell}?id=${video.id}"><input type="button" value="Excluir" style="ma
</label>
</form>
</div>
```

Segue acima, exibição dos campos do video, para atualização.

CHAPTER 8

Exclusão

Arquivo __init__.py

```
def videos_include(config):
    config.add_route('exclusao', '/excluir')
```

Configuração de rota para 'videos/excluir'

Arquivo views.py

```
@view_config(route_name='exclusao')
    def excluir(request):
    if request.params:
        try:
            id=request.params['id']
            DBSession.delete(DBSession.query(Video).filter_by(id=id).first())
            return HTTPFound(location=request.route_url('consulta'))
            except Exception:
                return Response("ID INVALIDO")
    else:
        return Response("KD O ID?")
```

Se existir algum parâmetro 'id' na requisição, o video referenciado é excluido. Se não existir vídeo identificado pelo 'id', então, o sistema exibe a seguinte mensagem: 'ID INVALIDO'. Se não existir nenhum parâmetro 'id', então, é exibido a seguinte mensagem: "KD O ID?".

CHAPTER 9

Rodando a aplicação

Para rodar a aplicação, você deve acessar o diretório-raíz de seu ambiente virtual (No nosso exemplo, dentro da pasta egito). Aí, considerando que você também está usando o Centos 7, execute o seguinte comando:

bin/pserve tutorial/development.ini

Aparecerá uma saída semelhante á esta:

Starting server in PID 11533. serving on http://0.0.0.0:6543

A saída acima indica que a aplicação está acessível na porta 6543. Mas, porta de onde? Da máquina que está hospedando esta aplicação. Caso seja sua máquina local, então, para testar, é só digitar na sua barra de endereço: http://localhost:5432/videos/.

CHAPTER 10

Lidando com templates

O framework Pyramid utiliza as seguintes linguagens para template: TAL, METAL e Mako. Estas linguagens podem ser usadas em conjunto. No nosso exemplo, é usada apenas a linguagem TAL, para acessar o valor de algumas variáveis e realizar iterações em algumas listagens. Linguagem de template é a linguagem utilizada para programar em arquivos estáticos (em formato HTML), por exemplo.

CHAPTER 11

Referências

https://media.readthedocs.org/pdf/sqlalchemy/rel_1_0/sqlalchemy.pdf

http://docs.pylonsproject.org/en/latest/

https://www.owlfish.com/software/simpleTAL/tal-guide.html