Formation Django

Vincent Angladon, Vincent Duvert

Jeudi 21 octobre 2010

http://www.bde.enseeiht.fr/clubs/net7/supportFormations/django/2011

Django ?

- Un framework
  - Gestion des requêtes et des URL
  - Interface vers une base de données
  - Formulaires
  - Authentification des utilisateurs
  - ...
- ... mais pas un CMS

Django ?

- Un framework
  - Gestion des requêtes et des URL
  - Interface vers une base de données
  - Formulaires
  - Authentification des utilisateurs
  - ...
- ... mais pas un CMS

Django ?

- Un framework
  - Gestion des requêtes et des URL
  - Interface vers une base de données
  - Formulaires
  - Authentification des utilisateurs
  - ...
- ... mais pas un CMS

Django ?

- Un framework
  - Gestion des requêtes et des URL
  - Interface vers une base de données
  - Formulaires
  - Authentification des utilisateurs
  - ...
- ... mais pas un CMS

Django ?

- Un framework
  - Gestion des requêtes et des URL
  - Interface vers une base de données
  - Formulaires
  - Authentification des utilisateurs
  - ...
- ... mais pas un CMS

Django ?

- Un framework
  - Gestion des requêtes et des URL
  - Interface vers une base de données
  - Formulaires
  - Authentification des utilisateurs
  - ...
- ... mais pas un CMS

Django ?

- Un framework
  - Gestion des requêtes et des URL
  - Interface vers une base de données
  - Formulaires
  - Authentification des utilisateurs
  - ...
- ... mais pas un CMS

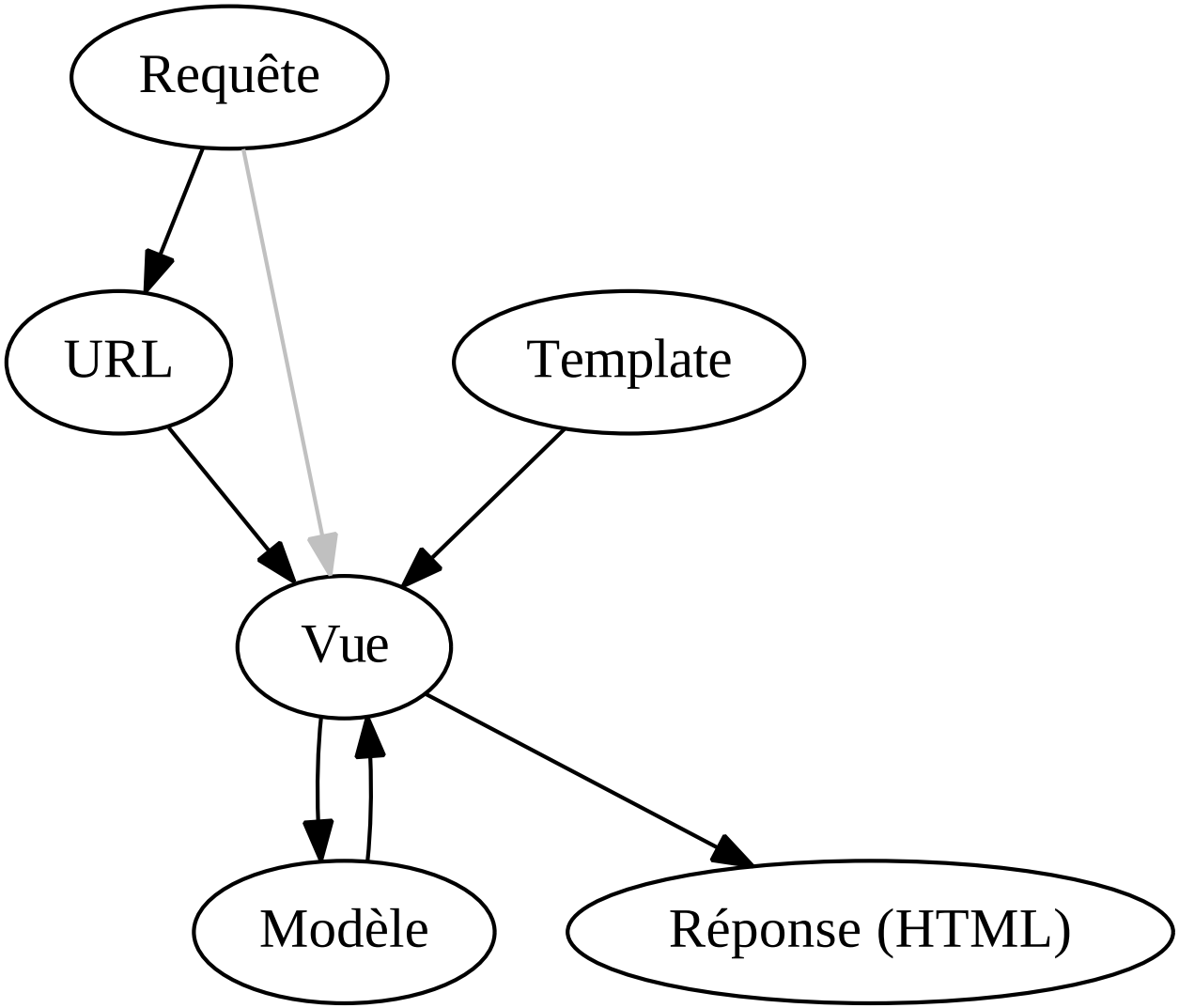
Django ?

- Un framework
  - Gestion des requêtes et des URL
  - Interface vers une base de données
  - Formulaires
  - Authentification des utilisateurs
  - ...
- ... mais pas un CMS

Attention

La version de Django actuelle 1.3.1 est basée sur Python2.x

Architecture



Créer un site Django

```
$ django-admin.py startproject monsite
$ ls monsite/
__init__.py manage.py   settings.py urls.py
$ cd monsite
$ python2 manage.py runserver
[...]
```

Votre 1e vue !

1. Créez un fichier views.py
2. Copiez y le code suivant :

```
# -*- coding: utf-8 -*-
from django.http import HttpResponse

def home(request):
    return HttpResponse("Hello")
```

La gestion des URLs

1. Ouvrez le fichier url.py
2. Copiez y le code suivant :

```
from django.conf.urls.defaults import patterns, \
                                      include, url


urlpatterns = patterns('',
      url(r'^$', 'monsite.views.home'),
)
```

page de référence sur les urls :
`https://docs.djangoproject.com/en/dev/topics/http/urls/`
page de référence sur les expressions régulières :
`http://docs.python.org/library/re.html`

URLs et expressions régulières

```
# ... suite de l'urlpattern dans urls.py
%(r'^annee/(\d{4})$', 'monsite.views.annee'),
%(r'^aff/(?P<texte>.+)/$', 'monsite.views.aff'),

# vues associees dans views.py
def annee(request, year):
    return HttpResponse("On est en "+year)

def aff(request, texte):
    return HttpResponse("Bonjour "+texte)
```

Exercice

Exercice : compléter monsite_exo/url.py

Solution

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

urlpatterns = patterns('',
    (r'^$', 'formations.views.accueil'),
    (r'formation/(?P<id>\d+)', \
        'formations.views.liste_inscrits'),
```

Première application

```
$ python2 manage.py startapp monapp
$ ls monapp
__init__.py  models.py  tests.py  views.py
```

1. ajouter l'application dans le tuple INSTALLED_APPS de settings.py
2. Déplacer la vue et les urls dans monapp
3. Modifier monsite/urls.py pour qu'il voit monsite/monapp/urls.py

```
# Rajouter cet urlpattern dans monsite/urls.py
(r'^',include('monapp.urls'))
```

Création d'un modèle

1. Configurer la base de donnée dans settings.py
2. Créer les modèles
3. Générer les tables à partir du modèle

Page de référence sur la configuration des bases de données :
`https://docs.djangoproject.com/en/dev/ref/settings/#databases`
Page de référence sur les champs des modèles :
`https://docs.djangoproject.com/en/dev/ref/models/fields/`

Premier modèle

```python
from django.db import models

class Article(models.Model):
    nom = models.CharField(verbose_name="Nom de l'article"\
            , max_length=128)
    description = models.TextField(max_length=512)
    prix = models.DecimalField(decimal_places=2, max_digits=6)
    tailles = models.TextField(blank = True)
    couleurs = models.TextField(default="blue")
    ecole = models.ForeignKey(Ecole, editable=False)

    def __unicode__(self):
        return "%s" % (self.nom)
```

Exercice

Exercice : compléter monsite_exo/formations/models.py

Solution

```python
# -*- coding: utf8 -*-
from django.db import models

class Formation(models.Model):
    nom = models.CharField(max_length=32)
    date = models.DateTimeField()

    def __unicode__(self):
        return self.nom

TYPE_INSCRIPTION = ((0, 'Inscrit'),
(1, 'Presentateur'),(2, 'Fictif'),)

class Inscription(models.Model):
    nom = models.CharField(max_length=32)
    type = models.SmallIntegerField(choices=TYPE_INSCRIPTION)
    formation = models.ForeignKey(Formation)

    def __unicode__(self):
        return u"%s (%s)" % (self.nom, self.get_type_display())
```

Création de la base de donnée

Création de la base :

```
$ python2 manage.py syncdb
```

Voir les commandes SQL exécutées :

```
$ python2 manage.py sql formations
```

Visualiser une base de donnée mysqlite :

```
$ sqlite3 mabase.sqlite
$ .tables
$ .schema formations_formation
$ select * from formations_formation;
```

Premières requêtes

```
$ python2 manage.py shell
```

Manipuler les modèles :

```
Modele.objects.all()
Modele.objects.get()
Modele.objects.filter()
entree.delete()
entree.save()
```

Pages de référence :

https://docs.djangoproject.com/en/1.3/topics/db/queries/
https://docs.djangoproject.com/en/1.3/ref/models/querysets/

Les Templates

Template = page html liée à une vue, qui a accès à certaines variables.

1. configurer TEMPLATE_DIRS dans settings.py
2. définir les variables affichées/utilisées par le template et les regrouper dans un dictionnaire
3. changer la vue pour quelle renvoie un render_to_response('montemplate.html', dictionnaire)

Pages de référence :

`https://docs.djangoproject.com/en/dev/ref/templates/api/`

`https://docs.djangoproject.com/en/dev/ref/templates/builtins/?`

Exemple d'utilisation

La vue

```
from django.template import RequestContext
from django.shortcuts import render_to_response

def accueil(request):
    c = {'formations': Formation.objects.all()}
    return render_to_response('accueil.html', c, \
        context_instance=RequestContext(request))
```

Le template accueil.html

```
<h1>Liste des formations</h1>
<ul>{% for f in formations %}
<li><a href="{% url formations.views.liste_inscrits f.id %}">
    {{ f }}</a></li>
{% endfor %}</ul>
```

Etendre des blocs

Page principale, main.html

```
<html lang="fr">
<head><title>{% block title %}{% endblock %}</title></head>
<body>
<h1>Bienvenue !</h1>
{% block content %}{% endblock %}
</body>
</html>
```

Page affichage.html qui va étendre main.html

```
{% extends "main.html" %}
{% block title %}Le titre de ma page{% endblock %}
{% block content %}
Du contenu ici.
{% endblock %}
```

Exercice

Exercice : compléter les fichiers dans monsite_exo/templates et les vues

Solution

```python
# -*- coding: utf8 -*-
from django.shortcuts import render_to_response\
     , get_object_or_404
from models import Formation

def accueil(request):
    c = {'formations': Formation.objects.all()}
    return render_to_response('accueil.html', c)

def liste_inscrits(request, id):
    formation = get_object_or_404(Formation, id=id)
    c = {
        'formation': formation,
        'liste': formation.inscription_set.all(),
    }
    return render_to_response('liste_inscrits.html', c)
```

Solution

accueil.html

```
{% extends "main.html" %}
{% block contents %}
<h1>Liste des formations</h1>
<ul>
{% for f in formations %}
<li><a href="{% url formations.views.liste_inscrits f.id %}">{
{% endfor %}
</ul>
{% endblock %}
```

Solution

liste_inscrits.html

```
{% extends "main.html" %}
{% block contents %}
<h1>{{ formation }}</h1>

<ul>
{% for i in formations %}
<li>{{ i }}</li>
{% empty %}
Aucun inscrit.
{% endfor %}
</ul>

{% endblock %}
```

A partir d'une modèle

```python
from django.forms import ModelForm
class MonModeleForm(ModelForm):
  class Meta:
    # Modele utilise pour generer le formulaire
    model = MonModele
    # champs du modele devant apparaitre dans le formulaire
    fields = ('champ1','champ2', 'champ3')
```

Son template, edit.html

```
<form action="" method="post">{% csrf_token %}
{{ form.as_p }}
<input type="submit" value="Submit" />
</form>
```

```
def edit(request, oid=None):
    if oid is not None:
        objet = MonModele.objects.get(id=oid)
    else:
        objet = None
    if request.method == 'POST':
        form = MonModeleForm(request.POST, instance=objet)
        if form.is_valid():
            form.save()
            return HttpResponseRedirect('/accueil')
    else:
        form = UtilisateurEditForm(instance=objet)
    return render_to_response('edit.html', {'form': form},\
     context_instance=RequestContext(request))
```

Interface d'administration

1. Décommenter les lignes concernant l'interface d'administration dans settings.py et urls.py
2. Mettre à jour les tables (syncdb)
3. Pour chaque application, indiquer les modèles pris en charge par l'interface d'administration via un fichier admin.py comme suit :

```
from monapp.models import *
from django.contrib import admin

admin.site.register(Modele1)
admin.site.register(Modele2)
```

Exercice : FaceMash

Cahier des charges, fonctions à implémenter :

- ajouter/éditer un utilisateur (Nom, adresse mail, photo) via un formulaire
- visualiser une liste des utilisateurs triée selon leur score
- visualiser le profil d'un utilisateur
- enregistrer les votes : date, vote, ip du votant (on est méchant)
- afficher la liste des votes
- vote : afficher les photos de 2 utilisateurs choisis aléatoirement, le vote s'effectue en cliquant sur l'une des photos
- interface d'administration