



Présentation ADRAR Pôle Numérique

Suivez-nous... LinkedIn  et Twitter  @Adrar_Numerique



ADRAR PÔLE NUMERIQUE

PÔLE NUMERIQUE DU CENTRE DE FORMATION ADRAR

- > SUPPORT, ADMINISTRATION SYSTEMES & RESEAUX
- > DEVELOPPEMENT D'APPLICATIONS WEB & MOBILES
- > WEBDESIGN & DEVELOPPEMENT INTERFACES
- > TRANSFORMATION NUMERIQUE DES ENTREPRISES

<http://www.adrar-numerique.com>

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com



Suivez-nous sur LinkedIn 

Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

TESTS

Suivez-nous sur LinkedIn  Twitter  **@Adrar_Numerique**

... et sur le web : **www.adrar-numerique.com**

Les tests font parties intégrantes d'un projet de développement. Encore une fois django fournit une série d'outils afin de faciliter la mise en place de tests. Afin de ne pas vous perdre dans la mise en place de vos tests je vous conseil de rédiger un test juste après avoir mis en place une fonctionnalité (ou juste avant si vous employez une approche de type TDD, test-driven development). Cependant pour ce cours nous allons rédiger les tests pour les fonctionnalités déjà présentes (mais représentez vous bien que si vous fonctionnez de cette manière sur un projet de grande envergure il est très facile de se perdre et d'oublier des tests).

Nous allons utiliser l'un des problèmes de notre code pour appréhender la notion de tests en django. En effet nous avons mis en place la méthode `was_published_recently()` retournant `True` si une question a été publié récemment. Le problème de notre méthode est qu'elle retourne `True` également si la question a une date de publication future.

Nous allons utiliser des tests pour corriger ce problème, mais tout d'abord voyons ca dans notre shell :

```
>>> from sondage.models import Question
>>> from django.utils import timezone
>>> import datetime
>>> futur_question = Question(pub_date = timezone.now() + datetime.timedelta(days=30))
>>> futur_question.was_published_recently()
True
```


Suivez-nous sur LinkedIn Twitter @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

Ce que nous venons de faire dans le shell est totalement une procédure à réaliser dans un test automatiser. Pour se faire nous allons rédiger un fichier *sondage/tests.py*. Django va automatiquement trouver les tests dans tous les fichiers dont le nom commence par *test*. Voyons ce que nous rédigeons dans ce fichier :

```
import datetime

from django.test import TestCase
from django.utils import timezone

from .models import Question

class QuestionModelTests(TestCase):

    def test_was_published_recently_with_future_question(self):
        """
        was_published_recently() returns False pour une question publiée dans le futur.
        """
        time = timezone.now() + datetime.timedelta(days=30)
        future_question = Question(pub_date=time)
        self.assertIs(future_question.was_published_recently(), False)
```

Nous créons dans ce fichier une classe *QuestionModelTests* héritant de la classe *TestCase*. Nous placerons tous les test de notre modèle Question dedans.

Nous définissons une méthode *test_was_published_recently_with_future_question()* en renseignant une docstring (représenté entre les triple guillemets) pour décrire son utilisation.

Nous commençons par créer un objet Question pour lequel la méthode *was_published_recently()* ne doit pas retourner *True*.

Nous utilisons ensuite la méthode *assertIs()* pour indiquer que l'appel de la méthode *was_published_recently()* sur l'objet généré auparavant doit retourner *False*.

Exécutons ensuite notre test. Pour cela nous allons une fois de plus utiliser le script *manage.py*.

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

```
PS C:\Users\maximeprzybylo\Documents\Cours\Django\TP\monsite> python .\manage.py test sondage
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
F
=====
FAIL: test_was_published_recently_with_future_question (sondage.tests.QuestionModelTests)
was_published_recently() returns False pour une question publiée dans le futur.
-----
Traceback (most recent call last):
  File "C:\Users\maximeprzybylo\Documents\Cours\Django\TP\monsite\sondage\tests.py", line 17, in test_was_published_recently_with_future_question
    self.assertIs(future_question.was_published_recently(), False)
AssertionError: True is not False
-----
Ran 1 test in 0.003s

FAILED (failures=1)
Destroying test database for alias 'default'...
```

Le script exécute notre test et nous indique qu'il a échoué en levant une exception *AssertionError* car la méthode a return *True* alors qu'elle aurait du return *False*.

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

Corrigeons donc notre méthode pour qu'elle passe notre test.

```
def was_published_recently(self):  
    now = timezone.now()  
    return now - datetime.timedelta(days=1) <= self.pub_date <= now
```

Cette fois notre méthode return True si la date de publication est comprise entre hier et aujourd'hui. Nous pouvons relancer notre test.

```
PS C:\Users\maximeprzybylo\Documents\Cours\Django\TP\monsite> python .\manage.py test sondage  
Creating test database for alias 'default'...  
System check identified no issues (0 silenced).  
.  
-----  
Ran 1 test in 0.001s  
  
OK
```

Suite à la modification de la méthode notre test est réussi.

Nous pouvons maintenant aller un peu plus loin dans le test de notre méthode.

Suivez-nous sur LinkedIn Twitter @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

```
def test_was_published_recently_with_old_question(self):
    """
    was_published_recently() returns False pour les questions plus vieilles qu'un jour.
    """
    time = timezone.now() - datetime.timedelta(days=1, seconds=1)
    old_question = Question(pub_date=time)
    self.assertIs(old_question.was_published_recently(), False)

def test_was_published_recently_with_recent_question(self):
    """
    was_published_recently() returns True pour les questions du jour.
    """
    time = timezone.now() - datetime.timedelta(hours=23, minutes=59, seconds=59)
    recent_question = Question(pub_date=time)
    self.assertIs(recent_question.was_published_recently(), True)
```

Nos deux nouvelles méthodes de tests vérifient que notre méthode `was_published_recently()` return bien la valeur souhaité en fonction de la date de publication :

False si la question est plus vieille d'un jour et d'une seconde

True si la question est bien d'aujourd'hui

Vue que nos méthodes de tests commencent toutes par `test`, django va toute les exécuter lors de l'appel du script `manage.py` avec l'argument `test sondage`

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

```
PS C:\Users\maximeprzybylo\Documents\Cours\Django\TP\monsite> python .\manage.py test sondage
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
...
-----
Ran 3 tests in 0.002s

OK
```

Django effectue donc nos 3 tests et nous dit que notre méthode renvoie bien la valeur souhaitée. Notre sondage est une application très limitée, c'est pourquoi nous n'avons que peu de tests mais pensez bien que plus le projet grandit plus vous aurez de tests (il n'est même pas incohérent d'avoir plus de lignes de tests que de lignes de codes). Même si cela vous paraît fastidieux ça reste un aspect essentiel d'un projet de développement qu'il faut faire de la manière la plus rigoureuse possible.

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

En plus de tester nos méthodes de modèles nous avons la possibilité de tester nos vues afin de s'assurer qu'elles nous affichent biens ce que l'on désire. On va commencer par voir nos possibilités par le shell interactif du script *manage.py*

```
>>> from django.test.utils import setup_test_environment
>>> setup_test_environment()
```

On commence par démarrer notre environnement de tests de gabarits.

```
>>> from django.test import Client
>>> client = Client()
```

On crée ensuite une instance de client pour tester l'accès à nos pages.

```
>>> response = client.get('/')
Not Found: /
>>> response.status_code
404
```

On effectue ensuite différents tests de réponse HTTP.

```
>>> from django.urls import reverse
>>> response = client.get(reverse('sondage:index'))
>>> response.status_code
200
>>> response.content
b'<!DOCTYPE html>\n<html>\n  <head>\n    <meta charset="utf-8">\n    <title>Mon sondage</title>\n    <link rel="stylesheet" type="text/css" href="/static/sondage/style.css">\n  </head>\n  <!-- H
TML de base -->\n  <body>\n    \n    <ul>\n      \n    <li>\n      <a href="/sondage/2/">Que voulez vous faire ?</a>\n    </li>\n    \n    <li>\n      <a href="/sondage/1/">Quel language de programmation pr\xc3\xa9f\xc3\xa9rez vous ?</a>\n    </li>\n    \n    </ul>\n    \n    </body>\n</html>
>\n\n'
>>> response.context['question_list']
<QuerySet [<Question: Que voulez vous faire ?>, <Question: Quel language de programmation préférez vous ?>]>
```

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

L'attribut *status_code* de notre objet *response* est un code de status HTTP. Voici une liste des codes erreurs les plus rependues (mais il en existe plein d'autre).

200 : succès de la requête

301 et 302 : redirection, respectivement permanente et temporaire

401 : utilisateur non authentifié

403 : accès refusé

404 : ressource non trouvée

500 et 503 : erreur serveur

504 : le serveur n'a pas répondu.

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

Modifions maintenant notre vue d'index pour s'assurer qu'elle n'affiche pas de question publié dans le futur.

```
class IndexView(generic.ListView):
    template_name = 'sondage/index.html'
    context_object_name = 'question_list'

    def get_queryset(self):
        return Question.objects.order_by('-pub_date')[:5]
```

```
def get_queryset(self):
    return Question.objects.filter(pub_date__lte=timezone.now()).order_by('-pub_date')[:5]
```

Nous modifions donc le retour de notre méthode `get_queryset()` pour y appliquer un filtre indiquant que la date de publication doit être inférieur ou égal (lte veut dire *less than or equal*) à la date du jours.

Nous pouvons ensuite rédiger notre tests pour s'assurer du bon fonctionnement.

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

Dans notre fichier de tests commençons par créer une fonctions qui permet de créer une question en lui passant comme paramètre le texte de la question et le nombre de jours à partir d'aujourd'hui qui nous séparent de la date de création (exemple : si on indique days=30, la date de publication sera dans 30 jours, si on indique days=-30 la date de publication sera il y a 30 jours).

```
def create_question(question_text, days):  
    time = timezone.now() + datetime.timedelta(days=days)  
    return Question.objects.create(question_text=question_text, pub_date=time)
```

On va ensuite créer une classe *QuestionIndexViewTest* héritant de *TestCase* afin d'y déclarer nos méthodes de test pour les différents cas : si il n'y a pas de question dans le sondage, si il n'y a que des questions futur, si il n'y qu'une question passé, si il y a une question passé et une question futur et si il y a plusieurs questions passées.

Voyons cette classe en détail :

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

```
class QuestionIndexViewTests(TestCase):
    def test_no_question(self):
        response = self.client.get(reverse('sondage:index'))
        self.assertEqual(response.status_code, 200)
        self.assertContains(response, "Pas de sondage disponible.")
        self.assertQuerysetEqual(response.context['question_list'], [])
```

Nos méthodes définissent toutes un objet *response* qui va être une simulation d'accès à notre page index par un client. Avec notre méthode *assertEqual* on test si le code de réponse de statut est bien 200. Avec *assertContains* on vérifie que, dans le cas où on a pas créé de question, la réponse contient bien « Pas de sondage disponible. » (car c'est ce que nous avons indiqué dans notre template index.html si il n'y a pas de question). Avec *assertQuerysetEqual* on vérifie que dans le context *question_list* il y a bien une liste vide

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

```
def test_past_question(self):
    question = create_question("Question passé", -10)
    response = self.client.get(reverse('sondage:index'))
    self.assertEqual(response.context['question_list'], [question])
```

```
def test_futur_question(self):
    create_question("Future question", 10)
    response = self.client.get(reverse('sondage:index'))
    self.assertContains(response, "Pas de sondage disponible.")
    self.assertEqual(response.context['question_list'], [])
```

Pour la méthode *test_past_question* nous commençons par créer une question donc la date de publication est antérieur à la date d'aujourd'hui.

Nous testons ensuite, via la méthode *assertQuerySetEqual* que dans le contexte *question_list* il y a bien la question créée.

Pour la méthode *test_futur_question* nous commençons par créer une question donc la date de publication est supérieur à la date d'aujourd'hui.

Nous testons ensuite, via la méthode *assertQuerySetEqual* que dans le contexte *question_list* il n'y a rien du tout.

Suivez-nous sur LinkedIn Twitter @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

```
def test_futur_and_past_question(self):
    question = create_question("Question passé", -2)
    create_question("Future question", 15)
    response = self.client.get(reverse('sondage:index'))
    self.assertQuerysetEqual(response.context['question_list'], [question])
```

```
def multiple_past_questions(self):
    question1 = create_question("Première question passé", -5)
    question2 = create_question("Deuxième question passé", -6)
    response = self.client.get(reverse('sondage:index'))
    self.assertQuerysetEqual(response.context['question_list'], [question1, question2])
```

Pour la méthode *test_futur_and_past_question* nous commençons par créer une question donc la date de publication est antérieur à la date d'aujourd'hui. Puis nous créons une question dont la date de publication est ultérieur à aujourd'hui. Nous testons ensuite, via la méthode *assertQuerysetEqual* que dans le contexte *question_list* il y a bien uniquement la question antérieur créée.

Pour la méthode *multiple_past_question* nous commençons par créer 2 question donc les dates de publication sont supérieur à la date d'aujourd'hui.

Nous testons ensuite, via la méthode *assertQuerysetEqual* que dans le contexte *question_list* il y a bien les 2 questions.

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

Procédons de la même manière pour la vue *DetailView*. Si on essaye d'accéder au détail d'une question dont la date de publication n'est pas encore arrivé, le site doit nous retourner un code erreur 404, sinon il nous affiche les informations de la question. Comme pour notre vue *IndexView*, notre vue *DetailView* doit d'abord appliquer un filtre sur le *QuerySet* passé dans notre contexte :

```
class DetailView(generic.DetailView):  
    model = Question  
    template_name = 'sondage/detail.html'  
  
    def get_queryset(self):  
        return Question.objects.filter(pub_date__lte=timezone.now())
```

Nous pouvons ensuite rédiger nos tests

Suivez-nous sur LinkedIn Twitter @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

```
class QuestionDetailViewTests(TestCase):
    def test_futur_question(self):
        futur_question = create_question("Futur question", 8)
        url = reverse('sondage:detail', args=(futur_question.id,))
        response = self.client.get(url)
        self.assertEqual(response.status_code, 404)

    def test_past_question(self):
        question = create_question("Question passé", -3)
        url = reverse('sondage:detail', args=(question.id,))
        response = self.client.get(url)
        self.assertContains(response, question.question_text)
```

Ici même logique que pour notre vue index, on crée des question en fonction de la situation et on teste le résultat de la requête.

Nous pouvons ensuite ré-exécuter nos tests pour nous assurer que tout va bien.

```
PS C:\Users\maximeprzybylo\Documents\Cours\Django\TP\monsite> python .\manage.py test sondage
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
-----
Ran 9 tests in 0.058s

OK
```