Réalisation de Test Unitaire Django

—

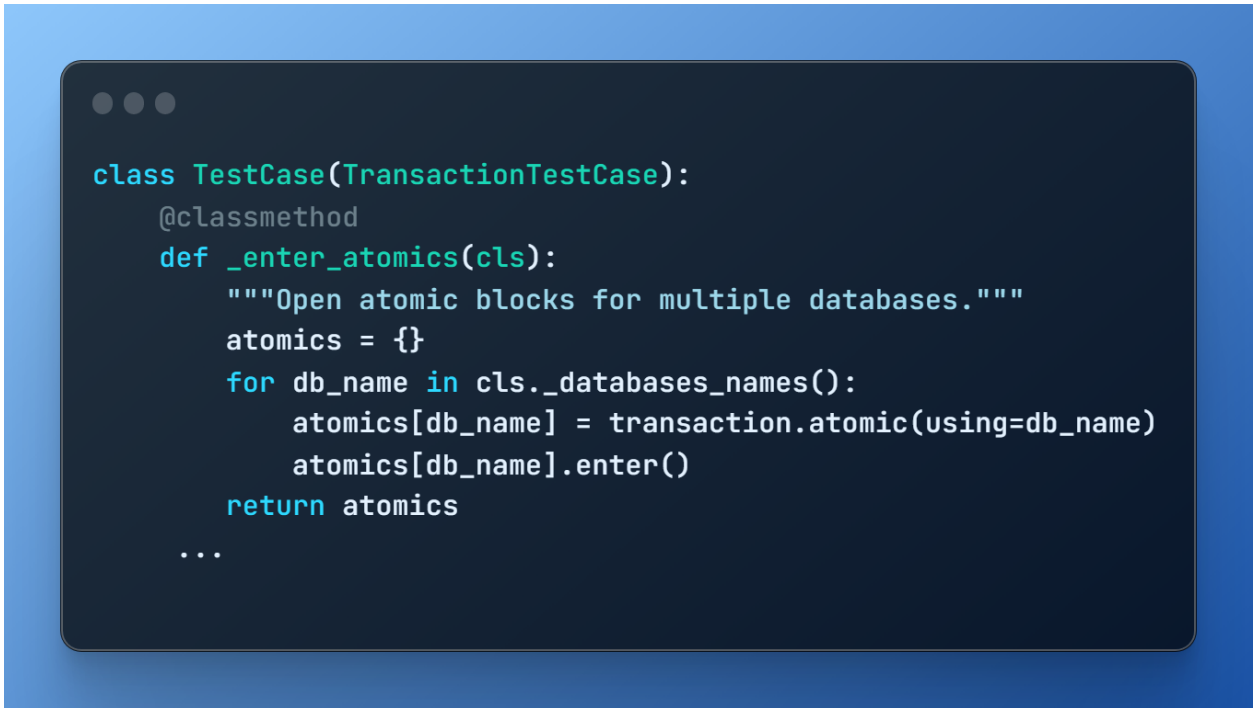
Yannis Alouache

Introduction

Le projet Django Obarbeuc est un projet important pour la mise en place d'une application Web fiable et robuste dans le cadre de la réservation de table de restauration.
Pour garantir la qualité et la fiabilité de l'application, l'utilisation des tests unitaires est indispensable.

Les Tests Unitaire

Dans le cadre de ce projet, la classe TestCase de Django a été utilisée pour créer des tests unitaires.



```python
class TestCase(TransactionTestCase):
    @classmethod
    def _enter_atomics(cls):
        """Open atomic blocks for multiple databases."""
        atomics = {}
        for db_name in cls._databases_names():
            atomics[db_name] = transaction.atomic(using=db_name)
            atomics[db_name].enter()
        return atomics
     ...
```

Un test est divisés en deux parties :

- SetUp : Initialise les méthodes et les données à tester
- Run : Lance le test et le compare avec le résultat attendu.

Un test est en fait une classe personnalisé composé de deux méthode qui hérite de la classe TestCase

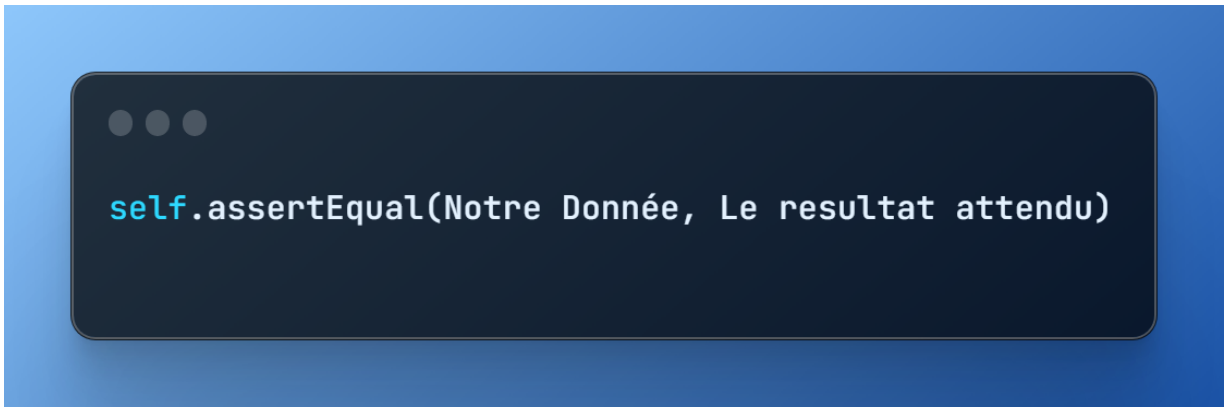
```python
class TodayLunchTestCase(TestCase):
    def setUp(self):
        TodayLunch.objects.create(title="Cheeseburger", price=35)
        TodayLunch.objects.create(title="SioBurger", price=25)
        TodayLunch.objects.create(title="BigBurger", price=10)


    def test_true_price(self):
        todayLunch_0 = TodayLunch.objects.get(title="Cheeseburger")
        self.assertEqual(todayLunch_0.getPrice(), 35)
    ....
```

Les tests ont été organisés dans un fichier nommé test.py.

La vérification des résultats a été effectuée à l'aide de la fonction assertEqual qui prend deux paramètres : la donnée et le résultat escompté.

Si la donnée est égale au résultat attendu le test est positif sinon on reçoit une notification sur ce qui à causer l'échec du test unitaire.



Dans le cadre de ce projet, les tests unitaires ont été utilisés pour tester les modèles et les urls de l'application. Cela permet de s'assurer que la logique derrière le code est appropriée et fonctionnera dans tous les cas.

Exemple d'un test qui vérifie que l'adresse mail d'une réservation

```
class ReservationTestCase(TestCase):
    def setUp(self):
        Reservation.objects.create(
          lastName="Alouache",
          firstName="Yannis",
          email="bigyanni1@gmail.com",
          phone="0685357448",
          date=datetime.date.today(),
          table=2,
          tableSetNumber=3,
          service="Midi"
        )

    def test_true_mail(self):
        reservation = Reservation.objects.get(lastName="Alouache")
        self.assertEqual(reservation.getEmail(), "bigyanni1@gmail.com")
```

Admettons que je lance ce test unitaire :

```python
class UrlTestCase(TestCase):
    def test_url(self):
        response = self.client.get('/reservation')
        self.assertEqual(response.status_code, 200)
```

J'essaye d'accéder à la page réservation sans être connecté ce qui va me bloquer et me renvoyer en code de redirection 302.

```
======================================================================
FAIL: test_url (main.tests.UrlTestCase)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\chill\Desktop\dev\OBarbeuc\obarbeuc\main\tests.py", line 12, in test_url
    self.assertEqual(response.status_code, 200)
AssertionError: 302 != 200

----------------------------------------------------------------------
Ran 1 test in 0.033s
```

Comme nous le voyons avec "AssertionError: 302 != 200"
La page nous a renvoyé le code 302 au lieu du code 200 qui était attendu.

Axe d’amélioration

Pour améliorer l'utilisation des tests unitaires dans le projet Obarbeuc,
J’aurais pu éventuellement ajouter des tests pour les vues et les formulaires.
Les tests de vues et de formulaires sont importants pour garantir le bon fonctionnement de l'application dans son ensemble.
J’aurais aussi pu faire plus de tests pour couvrir tous les aspects du code de l'application.