Uygulamalı Laboratuvar: Nesneleri Taklit Etme

Gerekli tahmini süre: 30 dakika

Nesneleri Taklit Etme laboratuvarına hoş geldiniz. Taklit etme, gerçek nesnelerin davranışını taklit eden nesneler oluşturma sürecidir. Bu, kodunuzun test sırasında mevcut olmayabilecek başka bir sisteme çağrıda bulunduğunda çok faydalı olabilir. Taklit etme, yalnızca kodunuzu test ettiğinizden emin olmak için kritiktir, başkalarının sistemini değil.

Bu laboratuvar sırasında, test sırasında Internet Movie Database (IMDb) hizmetine yapılan gerçek çağrıları taklit etmek için hem patching hem de mocking kullanacaksınız. Ayrıca, gerçekten çağırmış olsaydınız IMDb hizmetinden alacağınız geçerli yanıtları sağlamak için test fixture'ları kullanacaksınız. Bu şekilde, hizmete hiç çağrı yapmadan neyin döneceğini kontrol edebilirsiniz.

Öğrenme Hedefleri

Bu laboratuvarı tamamladıktan sonra şunları yapabileceksiniz:

- Test fixture'ları kullanarak gerçek API'lerden taklit yanıtlar yüklemek
- Test sırasında patch dekoratörünü kullanmak
- Mock sınıfını kullanarak diğer nesnelerin davranışını taklit etmek
- Test fixture verilerini kullanarak dönen nesneleri patch ve mock yapan test durumları yazmak

Theia Hakkında

Theia, masaüstü veya bulutta çalıştırılabilen açık kaynaklı bir IDE (Entegre Geliştirme Ortamı)dir. Bu laboratuvarı yapmak için Theia IDE'sini kullanacaksınız. Theia ortamına giriş yaptığınızda, size özel olarak 'bulutta ayrılmış bir bilgisayar' sunulur. Bu, laboratuvarlar üzerinde çalıştığınız sürece sizin için mevcuttur. Çıkış yaptığınızda, bu 'bulutta ayrılmış bilgisayar' ve oluşturmuş olabileceğiniz dosyalar silinir. Bu nedenle, laboratuvarlarınızı tek bir oturumda tamamlamak iyi bir fikirdir. Laboratuvarın bir kısmını tamamlayıp daha sonra Theia laboratuvarına döndüğünüzde, baştan başlamanız gerekebilir. Tüm Theia laboratuvarlarınızı tamamlamak için zamanınız olduğunda çalışmayı planlayın.

Laboratuvar Ortamını Kurun

Laboratuvara başlamadan önce biraz hazırlık yapmanız gerekiyor.

Terminali Açın

Editördeki menüyü kullanarak bir terminal penceresi açın: Terminal > Yeni Terminal.

Terminalde, eğer zaten `/home/projects` klasöründe değilseniz, şimdi proje klasörünüze geçin.

```
cd /home/project
```

Kodu Klonlayın

Şimdi test etmeniz gereken kodu alın. Bunu yapmak için, git deposunu klonlamak üzere `git clone` komutunu kullanın:

```
git clone https://github.com/ibm-developer-skills-network/duwjx-tdd_bdd_PracticeCode.git
```

Laboratuvar Klasörüne Geçin

Depoyu klonladıktan sonra, laboratuvar dizinine geçin:

```
cd duwjx-tdd_bdd_PracticeCode/labs/06_mocking_objects
```

Python Bağımlılıklarını Yükle

Son hazırlık adımı, laboratuvar için gerekli Python paketlerini yüklemek üzere `pip` kullanmaktır:

```
python3.8 -m pip install -r requirements.txt
```

Artık laboratuvara başlamak için hazırsınız.

Opsiyonel

Eğer terminalde çalışmak zorlaşırsa çünkü komut istemi çok uzunsa, aşağıdaki komutu kullanarak istemi kısaltabilirsiniz:

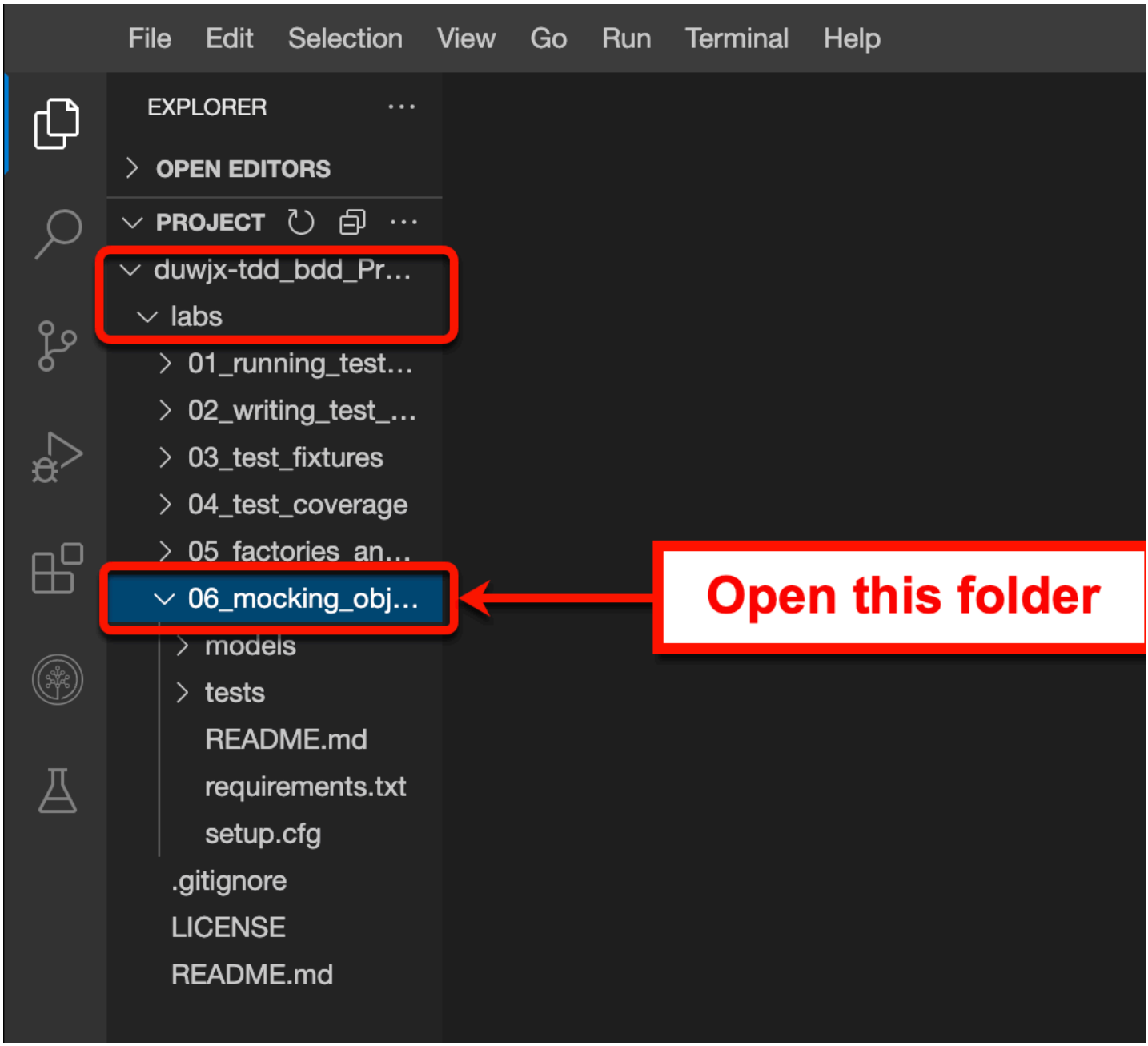
```
export PS1="[\[\033[01;32m\]\u\[\033[00m\]: \[\033[01;34m\]\W\[\033[00m\]]\$ "
```

Koda Git

Ekranınızın sağındaki IDE'de, `duwjx-tdd_bdd_PracticeCode/labs/06_mocking_objects` klasörüne gidin. Bu klasör, bu laboratuvar için kullanacağınız tüm kaynak kodunu içerir.

```
duwjx-tdd_bdd_PracticeCode
°
abs

    6·_mocking_objects
```



Test Düzeneklerini Kurma

`tests/fixtures/` klasöründe `imdb_responses.json` adlı bir dosya bulacaksınız. Bu dosya, IMDb API'sine çağrı yapılarak ve geri dönen yanıtlar kaydedilerek oluşturulmuştur. Her bir yanıt bir isim verilerek adlandırılmış ve test sırasında yüklenmek üzere bir `json` dosyasına yerleştirilmiştir.

Buna ek olarak, birkaç yanıt kopyalanmış ve iyi ve kötü yanıtları simüle etmek için değiştirilmiştir. Test koşulları altında neyin döneceğini kontrol edebilmenin ne kadar güçlü olduğunu görebilirsiniz. Yanıtları istediğiniz gibi şekillendirebilirsiniz.

Testlerde kullanacağınız çeşitli yanıtlarla tanışmak için IDE'de `tests/fixtures/imdb_responses.json` dosyasını açın.

Open **imdb_responses.json** in IDE

IMDb Sınıfı

`models/` klasöründe `imdb.py` adında bir dosya bulacaksınız. Bu modül, test edeceğiniz `IMDb` sınıfını içerir. Modül, IMDb hizmetinin sunduğu birçok API'den üç (3) tanesini uygulamaktadır. Özellikle, **SearchTitle**, **Reviews** ve **Ratings** API'leri sırasıyla `search_titles()`, `movie_reviews()` ve `movie_ratings()` yöntemleri tarafından uygulanmıştır.

Çeşitli yöntemlerle tanışmak için IDE'de `models/imdb.py` dosyasını açın. Bu yöntemleri testlerinizde çağıracaksınız.

Open **imdb.py** in IDE

Test Durumları

`tests/` klasöründe `test_imdb.py` adında bir dosya bulacaksınız. Bu, `IMDb` sınıfını test etmek için test durumlarınızı ekleyeceğiniz dosyadır.

IDE editöründe `tests/test_imdb.py` dosyasını açın. Laboratuvarın geri kalanında bu dosyada çalışacaksınız.

Open **test_imdb.py** in IDE

Adım 1: Başlığa Göre Arama Testi

Başlığa göre arama için bir test durumu uygulamaya başlayacaksınız. Aşağıda, şu anda herhangi bir yamanma veya taklit olmadan başlığa göre arama gerçekleştiren test yöntemi bulunmaktadır.

İlk Kod

Bu kodu `test_imdb.py` dosyasına ilk test olarak kopyalayıp yapıştırın ama henüz çalıştırmayın. Doğru bir şekilde girintilemeye dikkat edin:

```
    def test_search_by_title(self):
        """Test searching by title"""
        imdb = IMDb("k_12345678")
        results = imdb.search_titles("Bambi")
        self.assertIsNotNone(results)
        self.assertIsNone(results["errorMessage"])
        self.assertIsNotNone(results["results"])
        self.assertEqual(results["results"][0]["id"], "tt1375666")
```

Bu kod, bir API anahtarı ile başlatılan bir IMDb nesnesi oluşturur. Ardından kod, "Bambi" filmi için `imdb.search_titles()` çağrısını yapar ve sonuçların `None` olmadığını doğrular. Ayrıca hata mesajının boş olduğunu ve dönen `id`'nin `tt1375666` olduğunu kontrol eder.

Eğer gerçek bir IMDb API anahtarınız olsaydı, bu kod IMDb hizmetini çağırır ve bir yanıt dönerdi. Ancak, test sırasında API çağrılarınızın kotasını harcamak istemiyorsunuz, bu yüzden bu yöntemi `imdb.search_titles()`'ı hiç çağırmayacak şekilde patch'leyeceksiniz.

Göreviniz

`IMDb` sınıfının `search_titles()` yöntemini (yani, `IBDb.search_titles()`) hiç çağrılmayacak şekilde patch'lemek istiyorsunuz. Bunun için `@patch()` dekoratörünü kullanacak ve `return_value`'yu `GOOD_SEARCH` test fixture verilerini dönecek şekilde patch'leyeceksiniz.

1. `test_imdb.py` dosyasına, `test_search_by_title(self)` yönteminden önce aşağıdaki kod satırını ekleyin ve yeni mock için `imdb_mock` adında bir parametre ekleyin.

   ```
   @patch('test_imdb.IMDb.search_titles')
   def test_search_by_title(self, imdb_mock):
   ```

   Dikkat edin ki burada `test_imdb.IMDb.search_titles` üzerinde yamanma yapıyorsunuz. Test modülünüzün adı `test_imdb` ve bu nedenle, içe aktardığınız `IMDb` sınıfını yamanmalısınız, `models` paketindeki sınıfı değil. Bu kavramı anlamak önemlidir. Her zaman test ettiğiniz ad alanı içinde bulunan fonksiyonu yamamalısınız. Bu nedenle, `IMDb.search_titles`'ı `test_imdb.IMDb.search_titles` olarak tam olarak nitelendirmeniz gerekiyor.

2. Sonra, test yönteminin içinde, dokümantasyon dizesinden sonra ve `IMDb` sınıfını örneklendirme çağrısından önce, bu kod satırını ilk satır olarak ekleyin:

   ```
   imdb_mock.return_value = IMDB_DATA["GOOD_SEARCH"]
   ```

   `imdb_mock`'un, `@patch()` kullandıktan sonra yöntem çağrısına eklediğiniz ekstra parametre olduğunu unutmayın. Bu değişken, yapılan yamanın temsilcisidir. Bu değişken üzerinde `return_value` veya `side_effect` kullanabilirsiniz. Bu durumda, yamalanmış çağrıdan ne döneceğini kontrol etmek için `return_value` kullanıyorsunuz.

Bu iki değişiklik, `IMDb.search_titles()` yöntemini çağırmamak ve bunun yerine `GOOD_SEARCH` yanıtını döndürmek için yeterlidir.

Çözüm

▼ Çözüm için buraya tıklayın.

`test_imdb.py` dosyasında, `test_search_by_title(self)` yönteminin kodunu aşağıdaki kod ile değiştirin. Doğru şekilde girintilemeyi unutmayın.

```
    @patch('test_imdb.IMDb.search_titles')
    def test_search_by_title(self, imdb_mock):
        """Test searching by title"""
```

```
        imdb_mock.return_value = IMDB_DATA["GOOD_SEARCH"]
        imdb = IMDb("k_12345678")
        results = imdb.search_titles("Bambi")
        self.assertIsNotNone(results)
        self.assertIsNone(results["errorMessage"])
        self.assertIsNotNone(results["results"])
        self.assertEqual(results["results"][0]["id"], "tt1375666")
```

Testleri Çalıştır

`nosetests` komutunu çalıştırın ve test durumlarının geçtiğinden emin olun:

```
nosetests
```

Sonuçlar şöyle görünmelidir:

```
Tests Cases for IMDb Database
- Test searching by title
Name                Stmts   Miss  Cover   Missing
-------------------------------------------------
models/__init__.py      1      0   100%
models/imdb.py         24     15    38%   19-23, 27-31, 35-39
-------------------------------------------------
TOTAL                  25     15    40%
----------------------------------------------------------------------
Ran 1 test in 0.112s
OK
```

Adım 2: Sonuçsuz Arama

Artık yamanız ve taklit etmeniz gereken şeylerde yavaş yavaş daha sofistike hale geleceksiniz. Bu sonraki test bir "üzücü yol." Sonuç döndürmeyen bir çağrıyı test edecek.

İlk Kod

`test_search_with_no_results(self)` yönteminin yamanmamış versiyonunu `test_imdb.py` dosyasına kesip yapıştırarak başlayın. İşte kopyalamanız gereken kod:

```
        def test_search_with_no_results(self):
            """Test searching with no results"""
            imdb = IMDb("k_12345678")
            results = imdb.search_titles("Bambi")
            self.assertEqual(results, {})
```

Bu, bir API anahtarı ile yeni bir IMDb örneği oluşturur ve ardından `imdb.search_titles("Bambi")` çağrısını yapar ve geri dönenin boş bir sözlük olduğunu doğrular. IMDb hizmetinin başarısız olmasını sağlamadıkça bu pek olası değildir… ama bu hatayı bir mock ile simüle edebilirsiniz!

Göreviniz

1. `test_imdb.py` dosyasına, `test_search_with_no_results(self)` yönteminin öncesine aşağıdaki kod satırını ekleyin. Ardından yeni mock için `imdb_mock` adında bir parametre ekleyin. Bu kodun amacı, `requests.get()` çağrısını yamanlamak ve `imdb_mock` değişkenini kullanarak geri döneni kontrol etmektir.

```
@patch('models.imdb.requests.get')
def test_search_with_no_results(self, imdb_mock):
```

Bu sefer, `requests` adlı üçüncü taraf bir kütüphaneyi patch'lediğinizi unutmayın. Ancak, test modülünüze içe aktardığınız requests paketi değil. `imdb` modülündeki requests paketi (`models.imdb.requests.get`). Özellikle, `get` fonksiyonunu patch'liyorsunuz çünkü `IMDb.search_titles()` sonunda `requests.get()` yöntemini çağıracak ve IMDb API'sine istek yapacak. Bu isteği kesmek (veya patch'lemek) istiyorsunuz, böylece neyin döndüğünü kontrol edebilirsiniz.

2. Sonra, bu kod satırını test yönteminin içinde, docstring'den sonra ve `IMDb` sınıfını örneklendirme çağrısından önceki ilk satır olarak ekleyin:

```
imdb_mock.return_value = Mock(status_code=404)
```

Bu kodun, `requests.get()` çağrısının `return_value`'sını `status_code` özelliği `404` olarak ayarlanmış bir `Mock` nesnesi ile yamanacağını unutmayın. `IMDb.search_titles()` için kaynak kodunda arama yaparsanız, `requests.get()` çağrısı yapıldıktan sonra `status_code`'nun `200` olup olmadığını kontrol ettiğini göreceksiniz. Eğer durum kodu `200` değilse, kod boş bir sözlük `{}` döndürür. Test etmek istediğiniz davranış budur.

Bu iki değişiklik, `requests.get()` metodunun çağrılmamasını sağlamak ve bunun yerine `status_code`'su `404` olan bir `Mock` nesnesi döndürmek ve `{}` göndermek için yeterlidir.

Çözüm

▼ Çözüm için buraya tıklayın.

`test_imdb.py` dosyasında, `test_search_with_no_results(self)` metodu için şu anda sahip olduğunuz herhangi bir kodu aşağıdaki kod ile değiştirin. Doğru şekilde girintilemeyi unutmayın.

```
@patch('models.imdb.requests.get')
def test_search_with_no_results(self, imdb_mock):
    """Test searching with no results"""
    imdb_mock.return_value = Mock(status_code=404)
    imdb = IMDb("k_12345678")
    results = imdb.search_titles("Bambi")
    self.assertEqual(results, {})
```

Testleri Çalıştır

`nosetests` komutunu çalıştırın ve test durumlarının geçtiğinden emin olun:

```
nosetests
```

Sonuçlar şöyle görünmelidir:

```
Tests Cases for IMDb Database
- Test searching by title
- Test searching with no results
Name                 Stmts   Miss  Cover   Missing
--------------------------------------------------
models/__init__.py       1      0   100%
models/imdb.py          24     11    54%   22, 27-31, 35-39
--------------------------------------------------
TOTAL                   25     11    56%
----------------------------------------------------------------------
```

```
Ran 2 tests in 0.114s
OK
```

Adım 3: Başlıkla Arama Başarısız

Sonraki adımda, başka bir hata test durumu oluşturacaksınız, ancak bu sefer `requests` paketinden bir `Response` nesnesi gibi davranan bir mock oluşturmanız gerekiyor. İyi bir dönüş kodu olan `200` döndüreceksiniz ancak kötü bir API anahtarı kullanıyormuşsunuz gibi simülasyon yapmanız gerektiği için belirli bir hata mesajı döndürmeniz gerekiyor. Neyse ki, test verilerinizde `INVALID_API` adında bir hata mesajınız var.

Başlangıç Kodu

Başlamak için `test_search_by_title_failed(self)` metodunun yamanmamış versiyonunu `test_imdb.py` dosyasına kesip yapıştırın. İşte kopyalamanız gereken kod:

```
def test_search_by_title_failed(self):
    """Test searching by title failed"""
    imdb = IMDb("bad-key")
    results = imdb.search_titles("Bambi")
    self.assertIsNotNone(results)
    self.assertEqual(results["errorMessage"], "Invalid API Key")
```

Bu kodun, kötü bir API anahtarı geçerek yeni bir IMDb örneği oluşturduğunu unutmayın. Ardından kod, `imdb.search_titles("Bambi")` çağrısını yapar ve *"Geçersiz API Anahtarı"* hatası döndüğünü doğrular.

Göreviniz

1. `test_imdb.py` dosyasında, `test_search_by_title_failed(self)` yönteminden önce aşağıdaki kod satırını ekleyin ve yeni mock için `imdb_mock` adında bir parametre ekleyin. Bu kod, `requests.get()` çağrısına yapılan yamanmayı temsil eder. Bununla, `imdb_mock` değişkenini kullanarak ne döneceğini kontrol edebilirsiniz.

   ```
   @patch('models.imdb.requests.get')
   def test_search_by_title_failed(self, imdb_mock):
   ```

   Tekrar hatırlatmak gerekirse, `models` paketindeki `imdb` modülü tarafından içe aktarılan `requests` adlı üçüncü taraf kütüphanesini yamanıyorsunuz (yani `models.imdb.requests.get`). Özellikle, `get` fonksiyonunu yamıyorsunuz çünkü `IMDb.search_titles()` sonunda `requests.get()` metodunu çağırarak IMDb API'sine istek yapacak. O çağrıyı kesmek (veya yamak) istiyorsunuz ki geri döneni kontrol edebilesiniz.

2. İyi bir geri dönüş kodu olan `200` döneceksiniz. Bu geri dönüş kodu, `IMDb.search_titles()` metodunun dönen istekte `request.json()` çağrısını yapmasına neden olacak. `search_titles()` metodunun gerçek bir `requests.Response` aldığını düşünmesini sağlamak için, mock oluştururken `spec=Response,` kullanmalısınız, böylece gerçek `Response` sınıfı gibi davranır.

3. Ayrıca, istediğiniz yanıtı döndürmek için `json()` çağrısını da mock'lamanız gerekiyor: test verilerinizden `INVALID_API`. Bunu başarmak için, testinizden önce bir satır kod ekleyeceksiniz.

Sonraki adım, bu satırı test metodunun içindeki ilk satır olarak eklemektir. Dokstring'den sonra ancak `IMDb` sınıfını oluşturma çağrısından önce yerleştirin:

```
imdb_mock.return_value = Mock(
    spec=Response,
    status_code=200,
    json=Mock(return_value=IMDB_DATA["INVALID_API"])
)
```

Bu kodun, `requests.get()` çağrısının `return_value`'ını `status_code` özelliği `200` olarak ayarlanmış bir `Mock` nesnesi ile yamanmasına dikkat edin. `IMDb.search_titles()` kaynak kodunda arama yaparsanız, `requests.get()` çağrısı yapıldıktan sonra `status_code`'u kontrol ettiğini göreceksiniz. Eğer durum kodu `200` ise, ardından yükü almak için `request.json()` çağrısını yapar. Bu nedenle, `json()` çağrısını da taklit etmeniz ve istediğiniz yanıtı döndürmeniz gerekir.

Bu üç değişiklik, `requests.get()` yönteminin çağrılmamasını sağlamak ve bunun yerine `status_code`'u `200` olan bir `Mock` nesnesi döndürmek için yeterlidir. Ayrıca, çağrıldığında belirttiğiniz `INVALID_API` yanıtını döndüren bir `Response.json()` yöntemi de döndürecektir.

Çözüm

▼ Çözüm için buraya tıklayın.

`test_imdb.py` dosyasında, `test_search_by_title_failed(self)` yöntemi için şu anda sahip olduğunuz herhangi bir kodu aşağıdaki kod ile değiştirin. Doğru şekilde girintilemeyi unutmayın.

```
        @patch('models.imdb.requests.get')
        def test_search_by_title_failed(self, imdb_mock):
            """Test searching by title failed"""
            imdb_mock.return_value = Mock(
                spec=Response,
                status_code=200,
                json=Mock(return_value=IMDB_DATA["INVALID_API"])
            )
            imdb = IMDb("bad-key")
            results = imdb.search_titles("Bambi")
            self.assertIsNotNone(results)
            self.assertEqual(results["errorMessage"], "Invalid API Key")
```

Testleri Çalıştır

`nosetests` komutunu çalıştırın ve test durumlarının geçtiğinden emin olun:

```
nosetests
```

Sonuçlar şöyle görünmelidir:

```
Tests Cases for IMDb Database
- Test searching by title
- Test searching by title failed
- Test searching with no results
Name                 Stmts   Miss  Cover   Missing
--------------------------------------------------
models/__init__.py       1      0   100%
models/imdb.py          24     10    58%   27-31, 35-39
--------------------------------------------------
TOTAL                   25     10    60%
----------------------------------------------------------------------
Ran 3 tests in 0.111s
OK
```

Adım 4: Film Derecelendirmelerini Test Et

Bu son adımda, film derecelendirmeleri çağrısını test edeceksiniz. Gerçek IMDb veritabanını test sırasında çağırmak istemediğiniz için, `requests.get()` çağrısını bir kez daha mock'layacak ve test verilerinizden kendi film derecelendirme yanıtınızı yerine koyacaksınız.

Umarım uzaktan çağrıyı patch'leyerek, çağrıdan önce ve sonra fonksiyon kodunun geri kalanını test edebileceğinizi ve bunun her türlü test koşulunda düzgün çalıştığından emin olabileceğinizi görebiliyorsunuzdur.

Başlangıç Kodu

`test_movie_ratings(self)` metodunun patch'lenmemiş versiyonunu `test_imdb.py` dosyasına kopyalayıp yapıştırarak başlayın. İşte kopyalamanız gereken kod:

```
    def test_movie_ratings(self):
        """Test movie Ratings"""
        imdb = IMDb("k_12345678")
        results = imdb.movie_ratings("tt1375666")
        self.assertIsNotNone(results)
        self.assertEqual(results["title"], "Bambi")
        self.assertEqual(results["filmAffinity"], 3)
        self.assertEqual(results["rottenTomatoes"], 5)
```

Bu kodun, bir API anahtarı geçirerek yeni bir IMDb örneği oluşturduğunu unutmayın. Ardından, bir film kimliği geçirerek `imdb.movie_ratings({id})` çağrısını yapar. Son olarak, sonuçların `None` olmadığını doğrular ve doğru veriler olduğundan emin olmak için bazı derecelendirmeleri kontrol eder.

Göreviniz

1. `test_imdb.py` dosyasına, `test_movie_ratings(self)` yönteminden önce aşağıdaki kod satırını ekleyin ve yeni mock için `imdb_mock` adında bir parametre ekleyin. Bu kod, `requests.get()` çağrısına yapılan yamanmayı temsil eder. Bununla, `imdb_mock` değişkenini kullanarak ne döneceğini kontrol edebilirsiniz.

   ```
   @patch('models.imdb.requests.get')
   def test_movie_ratings(self, imdb_mock):
   ```

   Tekrar belirtmek gerekir ki, üçüncü taraf kütüphane fonksiyonu `requests.get()`'i patch'liyor ve `imdb_mock` adında bir değişken oluşturuyorsunuz. Bu değişken ile patch'in nasıl davranacağını kontrol edebilirsiniz.

2. İyi bir dönüş kodu olan `200`'ü geri göndereceksiniz. Bu dönüş kodu, `IMDb.movie_ratings()` metodunun dönen istekte `request.json()` çağrısı yapmasına neden olacaktır. `movie_ratings()`'ın gerçek bir `requests.Response` aldığını düşünmesi için, mock'u oluştururken `spec=Response,` kullanmalısınız ki gerçek `Response` sınıfı gibi davransın.

3. Yine, istediğiniz yanıtı döndürmesi için `json()` çağrısını mock'lamanız gerekiyor: test fixture verilerinizden `GOOD_RATING`. Bunu testinize bir satır kod ekleyerek başaracaksınız.

Sonraki adım, bu kod satırını test metodunun içindeki ilk satır olarak eklemektir. Bunu, docstring'den sonra ancak `IMDb` sınıfını örneklemek için yapılan çağrıdan önce yerleştirin:

```
    imdb_mock.return_value = Mock(
        spec=Response,
        status_code=200,
        json=Mock(return_value=IMDB_DATA["GOOD_RATING"])
    )
```

Dikkat edin, bu `requests.get()` çağrısının `status_code` özelliği `200` olarak ayarlanmış bir `Mock` nesnesi ile değiştirilmesidir. `IMDb.movie_ratings()` için kaynak kodunda arama yaparsanız, `requests.get()` çağrısı yapıldıktan sonra `status_code`'nun kontrol edildiğini göreceksiniz. Eğer `status code 200` ise, sonuçları almak için `request.json()` çağrılır. Bu nedenle, `json()` çağrısını da taklit etmeniz ve istediğiniz sonuçları döndürmeniz gerekir.

Bu üç değişiklik, `requests.get()` yönteminin çağrılmamasını sağlamak için yeterlidir ve bunun yerine `status_code`'su `200` olan bir `Mock` nesnesi döndürülür. Daha sonra `Response.json()` çağrıldığında, belirttiğiniz `GOOD_RATING` yanıtını geri gönderecektir.

Çözüm

▼ Çözüm için buraya tıklayın.

`test_imdb.py` dosyasında, `test_movie_ratings(self)` yöntemi için şu anda sahip olduğunuz kodu aşağıdaki kod ile değiştirin. Doğru bir şekilde girintilediğinizden emin olun.

```
    @patch('models.imdb.requests.get')
    def test_movie_ratings(self, imdb_mock):
        """Test movie Ratings"""
        imdb_mock.return_value = Mock(
            spec=Response,
            status_code=200,
            json=Mock(return_value=IMDB_DATA["GOOD_RATING"])
        )
        imdb = IMDb("k_12345678")
```

```
        results = imdb.movie_ratings("tt1375666")
        self.assertIsNotNone(results)
        self.assertEqual(results["title"], "Bambi")
        self.assertEqual(results["filmAffinity"], 3)
        self.assertEqual(results["rottenTomatoes"], 5)
```

Testleri Çalıştır

`nosetests` komutunu çalıştırın ve test durumlarının geçtiğinden emin olun:

```
nosetests
```

Sonuçlar şu şekilde görünmelidir:

```
Tests Cases for IMDb Database
- Test movie Ratings
- Test searching by title
- Test searching by title failed
- Test searching with no results
Name                 Stmts   Miss  Cover   Missing
----------------------------------------------------
models/__init__.py       1      0   100%
models/imdb.py          24      6    75%   27-31, 39
----------------------------------------------------
TOTAL                   25      6    76%
----------------------------------------------------------------------
Ran 4 tests in 0.109s
OK
```

Sonuç

Tebrikler! **Mock Objects** laboratuvarını tamamladınız. Umarım artık mock ile ilgili deseni tanıyorsunuzdur. Öncelikle, test durumunuzu bir patch ile sarmak için `@patch()` dekoratörünü kullanıyorsunuz. Özellikle, bu patch, test sırasında nihayetinde çağrılacak bir fonksiyon çağrısının davranışını değiştirecektir. Ardından, test yöntemi çağrısına, patched nesneyi temsil eden yeni bir parametre ekliyorsunuz. Son olarak, bu parametreyi kullanarak patched fonksiyonun davranışını değiştirecek `return_value` veya `side_effect`'i patch'liyorsunuz.

Ayrıca, `Response` sınıfı gibi diğer sınıfları taklit etmek için mock nesnelerini nasıl kullanacağınızı da öğrendiniz. Bu mock'lar ile bu sınıfların nasıl davrandığını ve ne döndüğünü kontrol edebilirsiniz. Hatta, mocked `Response` sınıfındaki `json()` fonksiyon çağrısını mock'layarak ne döndüğünü kontrol ettiniz.

Artık mock nesneleri oluşturma ve bunların nasıl davrandığını ve ne döndüğünü kontrol etme araçlarına sahipsiniz.

Bir sonraki zorluğunuz, bu teknikleri projelerinizde kullanarak test sırasında dış bağımlılıkları mock'lamak. Bunu yaparak, kodunuzun davranışını test ettiğinizden emin olabilirsiniz, başkasının hizmetini değil.

Author(s)

John J. Rofrano