Backend Testing with `unittest`, `Django`, and `Graphene`

Eric Campbell

Unit Testing with `unittest`

`unittest` Module — Setup

- Python comes with a built-in module for testing: `unittest`
- Several other alternatives, but this one is built in and does what we need
- Create a *test suite* by subclassing `unittest.TestCase` with a name starting with 'Test'
- Create a single test as a method of the test class with a name starting with 'test_' — your test won't be picked up without this

```
1 import unittest
2
3 class TestSomeStuff(unittest.TestCase):
4     test_a_thing(self):
5         assertions go here!
6     test_another_thing(self):
7         some more assertions!
```

Assertions

- General format in test:
  `self.assertSomething(params, "Message if the assertion fails")`
- Some useful ones:
  - `self.assertEqual(a, b, "message")` — whether the two are equal (==)
  - `self.assertTrue(complex condition, "message")` — whether some complex condition is true
  - `self.assertIn(elem, structure, "message")` — `elem in structure` — Very useful for testing GraphQL responses
  - `self.assertRaises(Exception, "message")` — it raises an exception of the given type (use with context provider)
- Also the usual suite of asserting greater than, not equal, is an instance of, &c.
- (Full list at the official docs)

Running

- To run it:

  ```
  python -m unittest test_some_stuff.py
  ```

- By default, the output is
  - Full stop: successful test
  - E: failed test
- Can set the verbosity with `--verbose` flag

Example — Testing an Addition Function i

Let's test the behaviour of our addition function:

add.py

```
1 # add.py
2 def add(a, b):
3     """Add two numbers"""
4     return a + b
```

It should:

1. Return numbers
2. Work correctly for `ints`
3. Work correctly for `floats`
4. Fail on anything else

Example — Testing an Addition Function ii

test_some_functions.py

```
1  import unittest
2
3  from add import add
4
5
6  class TestAdd(unittest.TestCase):
7      def test_returns_number(self):
8          """Should return a number"""
9          self.assertIsInstance(add(1, 2), (int, float),
               "Didn't return a number!")
10
11     def test_works_with_ints(self):
12         """Should correctly add integers"""
13         self.assertEqual(add(1, 2), 3, "NOOO!")
```

Example — Testing an Addition Function iii

```
14          self.assertEqual(add(-2, 2), 0)
15          self.assertEqual(add(-5, 100), 95, f":(")
16
17      def test_works_with_floats(self):
18          """Should correctly add floats"""
19          self.assertEqual(add(0.1, 0.2), 0.3, f"")
20          self.assertEqual(add(-0.2, 0.2), 0, f"")
21
22      def test_errors_on_non_numbers(self):
23          """When fed a non-number, it should raise a
                TypeError"""
24          bad_values = [["1", "2"], ["1", 2], [[], []],
                [{"hello": 3}, 5]]
25          for first, second in bad_values:
26              with self.assertRaises(TypeError):
```

Example — Testing an Addition Function iv

```
27 |                    add(first, second)
```

Example — Testing an Addition Function v

Output

```
$ python -m unittest test_some_functions.py

F.F.
======================================================================

FAIL: test_errors_on_non_numbers (test_some_functions.
    TestAdd)
When fed a non-number, it should raise a TypeError
----------------------------------------------------------------------

Traceback (most recent call last):
  File "/home/eric/documents/backend-testing-tutorial/
      simple-testing/test_some_functions.py", line 27,
      in test_errors_on_non_numbers
```

Example — Testing an Addition Function vi

```
10      add(first, second)
11  AssertionError: TypeError not raised
12
13  ======================================================================

14  FAIL: test_works_with_floats (test_some_functions.
        TestAdd)
15  Should correctly add floats
16  ----------------------------------------------------------------------

17  Traceback (most recent call last):
18    File "/home/eric/documents/backend-testing-tutorial/
          simple-testing/test_some_functions.py", line 19,
          in test_works_with_floats
19      self.assertEqual(add(0.1, 0.2), 0.3, f"")
```

Example — Testing an Addition Function vii

```
20 AssertionError: 0.30000000000000004 != 0.3 :
21
22 ----------------------------------------------------------------------
23 Ran 4 tests in 0.005s
24
25 FAILED (failures=2)
```

Discussion

Two things went wrong. One issue was with our function, and one was with the test.

1. Our function doesn't test for type — it needs to do that!
2. Our test (floats) didn't account for the fact that floating-point arithmetic is a bit dodgy. We should have used `self.assertAlmostEqual`

Example — Fixing Issues! i

Let's fix these issues!

Example — Fixing Issues! ii

add.py

```
1  # add.py
2  def add(a, b):
3      """Add two numbers"""
4      if not isinstance(a, (int, float)) or not
           isinstance(b, (int, float)):
5          raise TypeError(
6              "The␣inputs␣to␣the␣'add'␣function␣must␣be␣
                   numbers␣(int␣or␣float)!"
7          )
8
9      return a + b
```

Example — Fixing Issues! iii

test_some_functions.py

```
1  import unittest
2
3  from add import add
4
5
6  class TestAdd(unittest.TestCase):
7      def test_returns_number(self):
8          """Should return a number"""
9          self.assertIsInstance(add(1, 2), (int, float),
               "Didn't return a number!")
10
11     def test_works_with_ints(self):
12         """Should correctly add integers"""
13         self.assertEqual(add(1, 2), 3, "NOOO!")
```

Example — Fixing Issues! iv

```
14          self.assertEqual(add(-2, 2), 0)
15          self.assertEqual(add(-5, 100), 95, f":(")
16
17      def test_works_with_floats(self):
18          """Should correctly add floats"""
19          self.assertAlmostEqual(add(0.1, 0.2), 0.3)
20          self.assertAlmostEqual(add(-0.2, 0.2), 0)
21
22      def test_errors_on_non_numbers(self):
23          """When fed a non-number, it should raise a
                TypeError"""
24          bad_values = [["1", "2"], ["1", 2], [[], []],
                [{"hello": 3}, 5]]
25          for first, second in bad_values:
26              with self.assertRaises(TypeError):
```

Example — Fixing Issues! v

```
27 |                    add(first, second)
```

Example — Fixing Issues! vi

Output

```
$ python -m unittest --verbose test_some_functions.py

test_errors_on_non_numbers (test_some_functions.TestAdd
    )
When fed a non-number, it should raise a TypeError ...
    ok
test_returns_number (test_some_functions.TestAdd)
Should return a number ... ok
test_works_with_floats (test_some_functions.TestAdd)
Should correctly add floats ... ok
test_works_with_ints (test_some_functions.TestAdd)
Should correctly add integers ... ok

```

Example — Fixing Issues! vii

```
12 ----------------------------------------------------------------------

13 Ran 4 tests in 0.001s
14
15 OK
```

Your Turn

Take a few minutes and write your own test for the related `subtract.py`. Think about what sorts of behaviours you want, and try to avoid looking at the examples as much as possible.

Testing Django

Introduction

- `unittest` module is at the heart of testing Django
- For code that tests the actual database, Django provides a subclass of `unittest.TestCase` which you subclass for your tests
- `from django.test import TestCase`
- Will automatically start a clean DB instance at the beginning of *each* test and clean up when done
- You can freely mix 'regular' tests and Django ones in the same test suite
- To run these tests, need to use the 'test' command: `python manage.py test the_file_you_are_testing.py`
- You can also omit the filename to test all files it can find or specify a specific app

`setUp` and `tearDown`

- You will often want to have actions performed before and after each test in a suite (e.g. populate / clean the db)
- Define a `setUp` method: this will be run before each test in the suite
- Define a `tearDown` method: this will be run after each test in the suite
- (These are actually part of the base `unittest.TestCase`, so you can also use them in your regular tests)
- NB cleanup should be handled in `setUp` in case of a failing test

Example

```
1  from Django.test import TestCase
2  from app.models import * #need access to your models
3
4  class TestMyModel(TestCase):
5      def setUp(self):
6          populate the db here with fixtures
7      def test_model_works(self):
8          make assertions about models here (e.g. test for
               existence, changes, &c.)
```

The App

The application we're going to test is a simple listing of books.

models.py

```
1  from django.db import models
2
3  # Create your models here.
4  class Book(models.Model):
5      title = models.CharField(max_length=200)
6      author = models.ForeignKey("Author", on_delete=
           models.CASCADE, related_name="books")
7
8
9  class Author(models.Model):
10     first_name = models.CharField(max_length=200)
11     last_name = models.CharField(max_length=200)
```

Example — Testing `add_book_to_existing_author` i

```
1  from books.models import Book, Author
2
3
4  def add_book_to_existing_author(title, first_name,
       last_name) -> None:
5      """Adds a book to an existing author"""
6      author = Author.objects.get(first_name=first_name,
           last_name=last_name)
7      book = Book(title=title, author=author)
8      book.save()
```

Example — Testing `add_book_to_existing_author` ii

tests.py

```
1  from django.test import TestCase
2  from books.models import Book, Author
3  from books.utility import add_book_with_author,
       add_book_to_existing_author
4
5  # Create your tests here.
6  class TestAddBookToExistingAuthor(TestCase):
7      def setUp(self):
8          author1 = Author(first_name="Jack", last_name="
               London")
9          author2 = Author(first_name="Jack", last_name="
               Black")
10         author1.save()
11         author2.save()
```

Example — Testing `add_book_to_existing_author` iii

```
12          self.assertEqual(
13              len(Author.objects.all()),
14              2,
15              f"Wrong number of authors found in test
                    setup!",
16          )
17
18      def test_it_works(self) -> None:
19          """When given all parameters, should actually
                work"""
20          add_book_to_existing_author("White Fang", "Jack
                ", "London")
21
22          # the book sghould have been created
23          all_books = Book.objects.all()
```

Example — Testing `add_book_to_existing_author` iv

```
24            self.assertEqual(
25                len(all_books),
26                1,
27                f"Incorrect number of books — expecting
                      just the added one!",
28            )
29
30            #  the book should exist and have the right
                  title and author
31            white_fang = Book.objects.all()[0]
32            self.assertEqual(white_fang.title, "White Fang"
                  , f"Wrong title!")
33            self.assertEqual(
34                white_fang.author.first_name, "Jack", f"
                      Wrong Author first name!"
```

Example — Testing `add_book_to_existing_author` v

```
35            )
36            self.assertEqual(
37                white_fang.author.last_name, "London", f"
                      Wrong author last name!"
38            )
39
40            # also, they should be linked in the db
41            jack_london = Author.objects.get(first_name="
                  Jack", last_name="London")
42            self.assertEqual(
43                white_fang.author,
44                jack_london,
45                f"Book created, but not linked to the
                      existing author!",
46            )
```

Example — Testing `add_book_to_existing_author` vi

tests.py

```
1  $ python manage.py test books
2
3  Creating test database for alias 'default'...
4  .
5  ----------------------------------------------------------------------
6  Ran 1 test in 0.003s
7
8  OK
9  Destroying test database for alias 'default'...
10 System check identified no issues (0 silenced).
```

Example — Testing the Other One! i

`utility.py` contains another function to add a new book and a new author at the same time. Take a few moments and write some tests for it! Knowing that the signature is `add_book_with_author(title, first_name, last_name)`, try to write the tests *without* looking at `utility.py`

Example — Testing the Other One! ii

tests.py

```
1  from django.test import TestCase
2  from books.models import Book, Author
3  from books.utility import add_book_with_author,
       add_book_to_existing_author
4
5  # Create your tests here.
6  class TestAddBookToExistingAuthor(TestCase):
7      def setUp(self):
8          author1 = Author(first_name="Jack", last_name="
               London")
9          author2 = Author(first_name="Jack", last_name="
               Black")
10         author1.save()
11         author2.save()
```

Example — Testing the Other One! iii

```
12          self.assertEqual(
13              len(Author.objects.all()),
14              2,
15              f"Wrong number of authors found in test
                    setup!",
16          )
17
18      def test_it_works(self) -> None:
19          """When given all parameters, should actually
                work"""
20          add_book_to_existing_author("White Fang", "Jack
                ", "London")
21
22          # the book sghould have been created
23          all_books = Book.objects.all()
```

Example — Testing the Other One! iv

```
24            self.assertEqual(
25                len(all_books),
26                1,
27                f"Incorrect number of books — expecting
                      just the added one!",
28            )
29
30            #  the book should exist and have the right
                  title and author
31            white_fang = Book.objects.all()[0]
32            self.assertEqual(white_fang.title, "White Fang"
                  , f"Wrong title!")
33            self.assertEqual(
34                white_fang.author.first_name, "Jack", f"
                      Wrong Author first name!"
```

Example — Testing the Other One! v

```
35              )
36              self.assertEqual(
37                  white_fang.author.last_name, "London", f"
                        Wrong author last name!"
38              )
39
40              # also, they should be linked in the db
41              jack_london = Author.objects.get(first_name="
                    Jack", last_name="London")
42              self.assertEqual(
43                  white_fang.author,
44                  jack_london,
45                  f"Book created, but not linked to the
                        existing author!",
46              )
```

Example — Testing the Other One! vi

```
47
48
49  class TestAddBook(TestCase):
50      def test_should_work(self) -> None:
51          add_book_with_author("A Memoir", "Bob", "Howard
                ")
52
53          all_books = Book.objects.all()
54          all_authors = Author.objects.all()
55
56          # first check that they exist
57          self.assertEqual(
58              len(all_books), 1, f"Unexpected number of
                    books given that we only added 1!"
59          )
```

Example — Testing the Other One! vii

```
60              self.assertEqual(
61                  len(all_authors),
62                  1,
63                  f"Unexpected number of authors given that
                        we only added 1!",
64              )
65
66              # now check that they have the right attributes
67              the_book = all_books[0]
68              the_author = all_authors[0]
69
70              self.assertEqual(the_book.title, "A Memoir", f"
                    Wrong title!")
71              self.assertEqual(the_author.first_name, "Bob",
                    f"Wrong first name!")
```

Example — Testing the Other One! viii

```
72            self.assertEqual(the_author.last_name, "Howard"
                  , f"Wrong last name!")
73
74            # now check that they've been linked properly
75            self.assertEqual(
76                the_book.author, the_author, f"The book and
                      title weren't linked properly!"
77            )
```

Testing Graphene

Testing Graphene

- Graphene subclass `GraphQLTestCase`
- `from graphene_django.utils.testing import GraphQLTestCase`
- Allows you to make queries against the schema
- In the test:
  `self.query('graphql query / mutation', variables=variables)`
- Returns a response including the status `response.status` and a JSON of the content (`response.content`)

Differences

- Will need to import your schema
- Often useful to set up things common to an entire test suite using `setUpClass(cls)` — a class method
- Run once before ALL tests in the suite

Example — Sample Test i

```
1  from graphene_django.utils.testing import
       GraphQLTestCase
2  import json
3
4  from somewhere import the models you need
5  from somewhere.schema import schema
6
7  class TestSomeQueryOrWhatever(GraphQLTestCase):
8      GRAPHQL_SCHEMA = schema
9
10     @classmethod
11     def setUpClass(cls):
12         super(TestSomeQueryOrWhatever, cls).setUpClass()
13         cls.query_string = """
14         query {
```

Example — Sample Test ii

```
15              stuff {
16                  etc
17              }
18          }
19          """
20          now I generate the fixtures
21
22      def test_the_query(self):
23          query_variables = {'yes': False}
24          response = self.query(self.query_string,
                variables=query_variables)
25          decoded = json.loads(response.content)
26
27          assertions go here
```

The Setup

Here is an explanation of the Graphene side of things:

- Two queries:
  1. `allBooks`
  2. `allAuthors`
- Two mutations, roughly corresponding to the two functions we already looked at
  1. `createBookWithExistingAuthor`
  2. `createBookWithNewAuthor`

NB The two functions that we created need to be slighly modified to return the things they create

Graphene Files i

schema.py

```
1  import graphene
2  from graphene_django import DjangoObjectType
3  from books.models import Book, Author
4  from books.utility import add_book_to_existing_author,
       add_book_with_author
5
6
7  class BookType(DjangoObjectType):
8      class Meta:
9          model = Book
10         fields = ("title", "author")
11
12
```

Graphene Files ii

```
13 class AuthorType(DjangoObjectType):
14     class Meta:
15         model = Author
16         fields = ("first_name", "last_name")
17
18
19 class Query(graphene.ObjectType):
20     hello = graphene.String(default_value="Hi!")
21     all_books = graphene.List(BookType)
22     all_authors = graphene.List(AuthorType)
23
24     def resolve_all_books(root, info):
25         return Book.objects.all()
26
27     def resolve_all_authors(root, info):
```

Graphene Files iii

```
28          return Author.objects.all()
29
30
31  # mutations
32  class CreateBookWithExistingAuthorMutation(graphene.
        Mutation):
33      class Arguments:
34          title = graphene.String(required=True)
35          first_name = graphene.String(required=True)
36          last_name = graphene.String(required=True)
37
38      book = graphene.Field(BookType)
39      author = graphene.Field(AuthorType)
40
41      @classmethod
```

Graphene Files iv

```
42      def mutate(cls, root, info, **kwargs):
43          title = kwargs.get("title")
44          first_name = kwargs.get("first_name")
45          last_name = kwargs.get("last_name")
46          [book, author] = add_book_to_existing_author(
                title, first_name, last_name)
47          return CreateBookWithExistingAuthorMutation(
                book=book, author=author)
48
49
50  class CreateBookWithNewAuthorMutation(graphene.Mutation
        ):
51      class Arguments:
52          title = graphene.String(required=True)
53          first_name = graphene.String(required=True)
```

Graphene Files v

```
54          last_name = graphene.String(required=True)
55
56      book = graphene.Field(BookType)
57      author = graphene.Field(AuthorType)
58
59      @classmethod
60      def mutate(cls, root, info, **kwargs):
61          title = kwargs.get("title")
62          first_name = kwargs.get("first_name")
63          last_name = kwargs.get("last_name")
64          [book, author] = add_book_with_author(title,
                first_name, last_name)
65          return CreateBookWithNewAuthorMutation(book=
                book, author=author)
66
```

Graphene Files vi

```
67
68 class Mutation(graphene.ObjectType):
69     create_book_with_existing_author =
           CreateBookWithExistingAuthorMutation.Field()
70     create_book_with_new_author =
           CreateBookWithNewAuthorMutation.Field()
71
72
73 schema = graphene.Schema(query=Query, mutation=Mutation
       )
```

Graphene Files vii

utility.py

```
1  from books.models import Book, Author
2
3
4  def add_book_to_existing_author(title, first_name,
       last_name) -> None:
5      """Adds a book to an existing author"""
6      author = Author.objects.get(first_name=first_name,
           last_name=last_name)
7      book = Book(title=title, author=author)
8      book.save()
9      return [book, author]
10
11
```

Graphene Files viii

```
12 def add_book_with_author(title, first_name, last_name)
       -> None:
13     """Adds a book and author at the same time,
           relating them also"""
14     author = Author(first_name=first_name, last_name=
           last_name)
15     author.save()
16     book = Book(title=title, author=author)
17     book.save()
18     return [book, author]
```

Example — Testing Queries i

Let's test the `allBooks` query!

Example — Testing Queries ii

test_queries_and_mutations.py

```
1  from graphene_django.utils.testing import
       GraphQLTestCase
2  import json
3
4  from books.models import Book, Author
5  from backend_testing.schema import schema
6
7
8  class TestAllBooks(GraphQLTestCase):
9      GRAPHQL_SCHEMA = schema
10
11     @classmethod
12     def setUpClass(cls):
13         super(TestAllBooks, cls).setUpClass()
```

Example — Testing Queries iii

```
14          cls.query_string = """
15          query {
16              allBooks {
17                  title
18                  author {
19                      firstName
20                      lastName
21                  }
22              }
23          }
24          """
25
26          james_corey = Author(first_name="James",
                last_name="Corey")
27          james_corey.save()
```

Example — Testing Queries iv

```
28          gerald_durrell = Author(first_name="Gerald",
                last_name="Durrell")
29          gerald_durrell.save()
30
31          Book(title="Leviathan Wakes", author=
                james_corey).save()
32          Book(title="Caliban's War", author=james_corey)
                .save()
33          Book(title="The Bafut Beagles", author=
                gerald_durrell).save()
34          Book(title="My Family and Other Animals",
                author=gerald_durrell).save()
35
36      def test_all_books_works(self):
37          """It should return them all!"""
```

Example — Testing Queries v

```
38          response = self.query(
39              self.query_string,
40              variables={},
41          )
42          decoded = json.loads(response.content)
43
44          self.assertEqual(
45              response.status_code,
46              200,
47              f"Unexpected status code for syntactically
                    good query: {decoded}",
48          )
49          self.assertNotIn("error", decoded, f"Unexpected
                error in returned data")
```

Example — Testing Queries vi

```
50          self.assertIn("data", decoded, f"No data in
                response!")
51          self.assertIn("allBooks", decoded["data"], f"
                Returned data missing 'allBooks'")
52
53          allBooks = decoded["data"]["allBooks"]
54          self.assertEqual(len(allBooks), 4, f"Incorrect
                number of books returned!")
55          # &c.
```

Your Turn!

On your own, try to test the `allAuthors` query!

Example — Testing Mutations i

Luckily, testing mutations is essentially the same as queries — the main difference is that you also want to test that the changes to the database have taken as well! This means that we need to test the congruence between:

- The information we sent
- The information we received
- The information in the database

We also want to test that it fails in the way that we expect!

Let's test the `createBookWithExistingAuthor` mutation.

Example — Testing Mutations ii

test_queries_and_mutations.py

```
1  from graphene_django.utils.testing import
       GraphQLTestCase
2  import json
3
4  from books.models import Book, Author
5  from backend_testing.schema import schema
6
7
8  class TestAllBooks(GraphQLTestCase):
9      GRAPHQL_SCHEMA = schema
10
11     @classmethod
12     def setUpClass(cls):
13         super(TestAllBooks, cls).setUpClass()
```

Example — Testing Mutations iii

```
14          cls.query_string = """
15          query {
16              allBooks {
17                  title
18                  author {
19                      firstName
20                      lastName
21                  }
22              }
23          }
24          """
25
26          james_corey = Author(first_name="James",
                last_name="Corey")
27          james_corey.save()
```

Example — Testing Mutations iv

```
28          gerald_durrell = Author(first_name="Gerald",
                last_name="Durrell")
29          gerald_durrell.save()
30
31          Book(title="Leviathan Wakes", author=
                james_corey).save()
32          Book(title="Caliban's War", author=james_corey)
                .save()
33          Book(title="The Bafut Beagles", author=
                gerald_durrell).save()
34          Book(title="My Family and Other Animals",
                author=gerald_durrell).save()
35
36      def test_all_books_works(self):
37          """It should return them all!"""
```

Example — Testing Mutations v

```
38            response = self.query(
39                self.query_string,
40                variables={},
41            )
42            decoded = json.loads(response.content)
43
44            self.assertEqual(
45                response.status_code,
46                200,
47                f"Unexpected status code for syntactically
                      good query: {decoded}",
48            )
49            self.assertNotIn("error", decoded, f"Unexpected
                  error in returned data")
```

Example — Testing Mutations vi

```
50          self.assertIn("data", decoded, f"No data in
                response!")
51          self.assertIn("allBooks", decoded["data"], f"
                Returned data missing 'allBooks'")
52
53          allBooks = decoded["data"]["allBooks"]
54          self.assertEqual(len(allBooks), 4, f"Incorrect
                number of books returned!")
55          # &c.
56
57
58  class TestCreatBookWithExistingAuthor(GraphQLTestCase):
59      GRAPHQL_SCHEMA = schema
60
61      @classmethod
```

Example — Testing Mutations vii

```
62      def setUpClass(cls):
63          super(TestCreatBookWithExistingAuthor, cls).
                setUpClass()
64          cls.query_string = """
65              mutation createBookWithExistingAuthor(
                    $title: String!, $firstName: String!,
                    $lastName: String!) {
66                  createBookWithExistingAuthor(title:
                        $title, firstName: $firstName,
                        lastName: $lastName) {
67                      book {
68                          title
69                      }
70                      author {
71                          firstName
```

Example — Testing Mutations viii

```
72                                    lastName
73                                }
74                            }
75                        }
76                            """
77
78            # now create the author we will be adding to
79            Author(first_name="Robert", last_name="Martin")
                  .save()
80
81        def test_fail_with_missing_params(self) -> None:
82            """Missing parameters -> fail"""
83            query_variables = {
84                "firstName": "Robert",
85                "lastName": "Martin",
```

Example — Testing Mutations ix

```
86              }
87              response = self.query(self.query_string,
                    variables=query_variables)
88              decoded = json.loads(response.content)
89
90              self.assertEqual(
91                  response.status_code,
92                  400,
93                  f"Unexpected status code for syntactically
                        bad query: {decoded}",
94              )
95              self.assertIn(
96                  "errors",
97                  decoded,
```

Example — Testing Mutations x

```
98                f"Unexpected lack of 'errors' key in
                      response to bad query: {decoded}",
99            )
100
101       def test_fail_with_missing_author(self) -> None:
102           """Author doesn't exist -> fail"""
103           query_variables = {
104               "firstName": "Some",
105               "lastName": "Guy",
106               "title": "Clean Code",
107           }
108           response = self.query(self.query_string,
                  variables=query_variables)
109           decoded = json.loads(response.content)
110
```

Example — Testing Mutations xi

```
111             self.assertEqual(
112                 response.status_code,
113                 200,
114                 f"Unexpected status code for syntactically
                        good query: {decoded}",
115             )
116             self.assertIn(
117                 "errors",
118                 decoded,
119                 f"Unexpected lack of 'errors' key in
                        response to bad query: {decoded}",
120             )
121
```

Example — Testing Mutations xii

```
122     def
            test_create_book_with_existing_author_should_work
            (self) -> None:
123         """When given correct parameters, the query
                should work"""
124         query_variables = {
125             "title": "Clean Code",
126             "firstName": "Robert",
127             "lastName": "Martin",
128         }
129         response = self.query(self.query_string,
                variables=query_variables)
130         decoded = json.loads(response.content)
131
```

Example — Testing Mutations xiii

```
132             self.assertIn("data", decoded, f"Response
                    missing data: {decoded}")
133             self.assertIn(
134                 "createBookWithExistingAuthor",
135                 decoded["data"],
136                 f"Returned data missing '
                        createBooksWithExistingAuthor ': {
                        decoded['data']}",
137             )
138             self.assertIn(
139                 "book",
140                 decoded["data"]["
                        createBookWithExistingAuthor"],
```

Example — Testing Mutations xiv

```
141                 f"Returned data missing 'book': {decoded['
                        data']['createBookWithExistingAuthor']}
                        ",
142             )
143             book = decoded["data"]["
                    createBookWithExistingAuthor"]["book"]
144             self.assertIn(
145                 "author",
146                 decoded["data"]["
                        createBookWithExistingAuthor"],
147                 f"Returned data missing 'author': {decoded
                        ['data']['createBookWithExistingAuthor
                        ']}",
148             )
```

Example — Testing Mutations xv

```
149             author = decoded["data"]["
                    createBookWithExistingAuthor"]["author"]
150
151             # now test that what was received is what was
                    sent
152
153             self.assertEqual(
154                 book,
155                 {"title": query_variables["title"]},
156                 f"Returned book was not what we expected: {
                        query_variables}; {book}",
157             )
158             self.assertEqual(
159                 author,
160                 {
```

Example — Testing Mutations xvi

```
161                     "firstName": query_variables["firstName
                            "],
162                     "lastName": query_variables["lastName"
                            ],
163                 },
164                 f"Returned author was not what was expected
                        ! { query_variables }; { author }",
165             )
166
167             # now test that what was created in the
                    database is what we sent / received
168             book_in_db = Book.objects.get(title=
                    query_variables["title"])
169             author_in_db = book_in_db.author
170
```

Example — Testing Mutations xvii

```
171             self.assertEqual(
172                 book_in_db.title,
173                 book["title"],
174                 f"Received book and that in the databse do
                        not match! {book}; {book_in_db}",
175             )
176             self.assertEqual(
177                 author_in_db.first_name,
178                 author["firstName"],
179                 f"First name of the received author and the
                        one in the DB do not match: {
                        author_in_db}; {author}",
180             )
181             self.assertEqual(
182                 author_in_db.last_name,
```

Example — Testing Mutations xviii

```
183                 author["lastName"],
184                 f"First name of the received author and the
                        one in the DB do not match: {
                        author_in_db}; {author}",
185             )
```

Important

- Test for the status *and* the presence of an `error` attribute!
  - Status 200: query is syntactically good!
  - Status 400: query is syntactically invalid!
  - `error` attribute: error once the query has been delivered (i.e. in the resolver)
  - Careful: a bad response will still have a `data` attribute, but now this will contain data about the error
- When testing failure modes, be very careful that it is failing for the reason you think! Generally, start with a valid query and go from there
- For queries, test database $\iff$ `received`
- For mutations, test database $\iff$ sent $\iff$ received

Your Turn!

On your own, try testing the `createBookWithNewAuthor` mutation!

THE END

!