Django testing

Using pytest and Hypothesis for test generation

What we will cover during this presentation

- Pytest
- Pytest-django plugin
- Using pytest parametrization to generate tests
- Adding hypothesis into the mix

What to expect

Beginners welcome!

- Since we are focusing on pytest and hypothesis, you can use this techniques for other frameworks or testing in general
- We won't be going too much in depth, but hopefully you will have enough knowledge to start using it in your projects

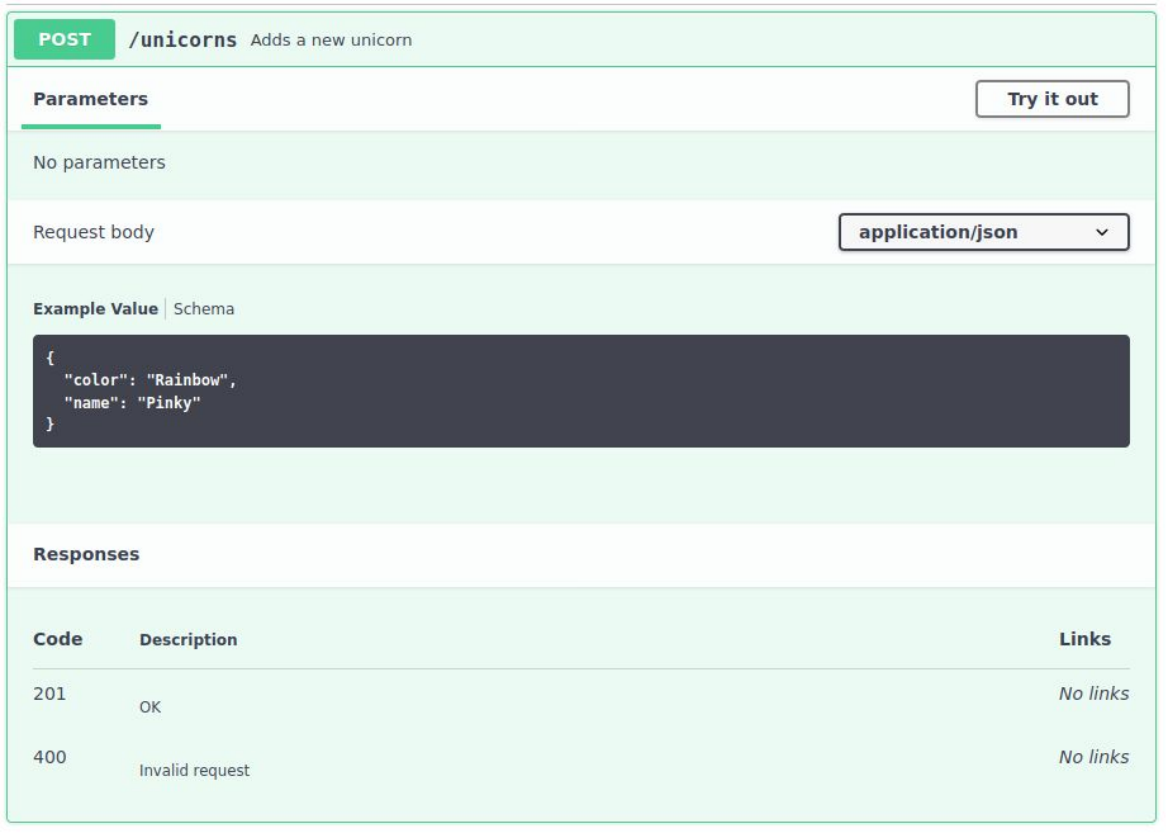
What will we do

- Make a website for keeping track of unicorns
- Add new unicorn and list added unicorns

Before testing

- Know what are you building
- Play around with low fidelity prototypes aka get paper and start drawing
- Write an API specification
- Sleep
- Rewrite API specification

Our first end-point



Our Unicorn model

```
class Unicorn(models.Model):
    class Colors(models.TextChoices):
        RAINBOW = 'Rainbow'
        DOUBLE_RAINBOW = 'Double rainbow'
        SUPER_RAINBOW = 'Super rainbow'

    name = models.CharField(max_length=30, validators=[MinLengthValidator(limit_value=2)])
    color = models.CharField(choices=Colors.choices, max_length=30)
```

Couple of test to make sure unicorns are ok

```python
def test_add_unicorn_empty_data(client):
    response = client.post('/unicorns/', data={})
    assert response.status_code == 400


def test_add_unicorn_valid_data(client, db):
    valid_request = {
        'name': 'Shiny',
        'color': Unicorn.Colors.RAINBOW
    }
    response = client.post('/unicorns/', valid_request)
    assert response.status_code == 201
    unicorn = Unicorn.objects.first()

    assert unicorn is not None
    assert unicorn.name == valid_request['name']
    assert unicorn.color == valid_request['color']
```

Our simple view up to this point

```
class UnicornsView(TemplateView):
    def post(self, request, *args, **kwargs):
        if request.POST == {}:
            return HttpResponseBadRequest()
        unicorn = Unicorn(name=request.POST['name'], color=request.POST['color'])
        unicorn.save()
        return JsonResponse(data={'name': unicorn.name,
                                  'color': unicorn.color}, status=201)
```

Sending invalid requests structure

Two tests for the price of one

```python
@pytest.mark.parametrize('key', ['name', 'color'])
def test_invalid_request(client, key):
    valid_request = {
        'name': 'Shiny',
        'color': Unicorn.Colors.RAINBOW
    }
    del valid_request[key]
    response = client.post('/unicorns/', valid_request)
    assert response.status_code == 400
```

Sending invalid request data types

```python
@pytest.mark.parametrize('name', [1, True, -0.5])
def test_invalid_request(client, name):
    valid_request = {
        'name': name,
        'color': Unicorn.Colors.RAINBOW
    }
    response = client.post('/unicorns/', valid_request)
    assert response.status_code == 400
```

Enter the Hypothesis

- Rewriting above tests using hypothesis
- Benefits of testing all sorts of crazy inputs (unicode chaos)

Type to test valid text inputs with hypothesis

Hundred tests with one simple fixture

```
@given(name=st.text(min_size=2, max_size=30))
def test_various_name_inputs(client, db, name):
    valid_request = {
        'name': name,
        'color': Unicorn.Colors.RAINBOW
    }
    response = client.post('/unicorns/', valid_request)
    assert response.status_code == 201
```

Contact information

Connect with me on linkedin and let's chat about python

https://www.linkedin.com/in/boyan-miletic/

Email: bojan@softerrific.com

Questions?