Django test

Unittest python com alguns diferenciais

Django Test

INTRODUCAO

- Django oferece ferramentas nativas para criar, gerenciar e executar testes automatizados.

- Baseado em unittest (padrão do Python), mas com recursos extras para trabalhar com:
  - Banco de dados de teste
  - Simulação de requisições HTTP
  - Validação de respostas de API

Tipos de testes

- Testes Unitários
- Testes de integração
- Testes Funcionais (E2E)

Arquitetura

```
project/
│
├── myapp/
│   ├── models.py
│   ├── views.py
│   ├── serializers.py
│   └── tests/
│       ├── test_models.py
│       ├── test_views.py
│       ├── test_serializers.py
│       ├── test_urls.py
│       └── test_api.py
```

Comandos principais

python manage.py test	Executa todos os testes
python manage.py test app.tests	Testa apenas um app específico
python manage.py test app.tests.test_models	Testa apenas um arquivo específico

```python
class UsuarioModelTest(TestCase):

    def test_create_usuario(self):
        user = usuario.objects.create(
            email = 'teste@teste.com',
            nome = 'teste',
            sobrenome = 'testando',
            is_active = True,
            is_staff = False,
            tipo = 'locatario'
        )

        self.assertEqual(user.email, 'teste@teste.com')
        self.assertEqual(user.nome, 'teste')
        self.assertEqual(user.sobrenome, 'testando')
        self.assertEqual(user.is_active, True)
        self.assertEqual(user.is_staff, False)
        self.assertEqual(user.tipo, 'locatario')
```

```python
def test_email_unico(self):

        usuario.objects.create(
                email = 'teste@teste.com',
                nome = 'teste',
                sobrenome = 'testando',
                is_active = True,
                is_staff = False,
                tipo = 'locatario'
                )

        with self.assertRaises(IntegrityError):
                user2 = usuario.objects.create(
                email = 'teste@teste.com',
                nome = 'teste2',
                sobrenome = 'testando2',
                is_active = True,
                is_staff = False,
                tipo = 'locatario'
                )
```

Juntando com o Django Rest Framework, podemos criar testes mais elaborados

- Testes de rotas protegidas
- Teste de cookies e headers
- Testes de tokens de autenticacao

Cliente HTTP para testar requisições web

APIClient – O Cliente de Teste de Alto Nível

Um cliente HTTP sofisticado projetado especificamente para testar APIs REST.
Principais características:

- Simula requisições HTTP completas (GET, POST, PUT, DELETE, etc.)
- Lida automaticamente com:
  - Autenticação (JWT, tokens, sessões)
  - Content-Type (JSON, form-data)
  - Cookies e headers
- Integração direta com o sistema de rotas do Django

```python
class LoginViewTests(TestCase):
    def setUp(self):
        self.client = APIClient()
        self.url = '/api/entrar/'
        self.user = User.objects.create_user(
            email='test@example.com',
            nome='Test User',
            password='testpassword123',
            tipo='proprietario'
        )
        self.valid_data = {
            'email': 'test@example.com',
            'password': 'testpassword123'
        }
```

```python
def test_login_sucesso(self):
    response = self.client.post(self.url, self.valid_data)
    self.assertEqual(response.status_code, status.HTTP_200_OK)


    self.assertEqual(response.data['email'], 'test@example.com')
    self.assertEqual(response.data['nome'], 'Test User')


    self.assertIn('access_token', response.cookies)
    self.assertIn('refresh_token', response.cookies)


    refresh_token = response.cookies['refresh_token'].value

    self.assertTrue(RefreshToken(refresh_token))
    self.assertTrue(RefreshToken(refresh_token).access_token)
```

APITestCase

APITestCase é uma classe de teste do Django REST Framework (DRF) usada para testar APIs REST de ponta a ponta com suporte completo a requisições HTTP, autenticação, e validações de resposta.

```python
class JWTTests(APITestCase):

    def setUp(self):
        self.user = User.objects.create_user(
            email='jwtuser@test.com',
            password='jwtpassword123',
        )
        self.refresh = RefreshToken.for_user(self.user)
        self.access_token = str(self.refresh.access_token)
        self.url_rota_protegida = '/api/me/'

    def test_acesso_com_token_valido(self):
        self.client.cookies['access_token'] = str(self.access_token)
        self.client.credentials(HTTP_AUTHORIZATION=f'Bearer {self.access_token}')

        response = self.client.get(self.url_rota_protegida)



        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertEqual(response.data['user']['email'], self.user.email)
        self.assertEqual(response.data['user']['nome'], self.user.nome)
```

```python
def test_acesso_sem_token(self):
    response = self.client.get(self.url_rota_protegida)

    self.assertEqual(response.status_code, status.HTTP_401_UNAUTHORIZED)
    self.assertIn('detail', response.data)

def test_acesso_com_token_invalido(self):
    self.client.credentials(HTTP_AUTHORIZATION='Bearer token-invalido')

    response = self.client.get(self.url_rota_protegida)

    self.assertEqual(response.status_code, status.HTTP_401_UNAUTHORIZED)
    self.assertIn('detail', response.data)
```

Ferramentas extras

FACTORY BOY

- O Factory Boy é uma biblioteca Python usada para criar objetos de teste de forma automática e consistente, especialmente útil em testes unitários e de integração, principalmente com ORMs como Django ORM, SQLAlchemy, etc.

- Ele se inspira no conceito de "factories" do Ruby on Rails (FactoryBot) e é projetado para substituir a prática de criar objetos manualmente nos testes, oferecendo mais legibilidade, reuso e controle.

```python
class PessoaFactory(factory.django.DjangoModelFactory):
    class Meta:
        model = usuario

    nome = "Joao da Silva"
    email = "orlando@teste.com"



class UsuarioModelTest(TestCase):

      def test_criacao_de_pessoa1(self):
            pessoa = PessoaFactory()

            self.assertEqual(pessoa.email, 'orlando@teste.com')
            self.assertEqual(pessoa.nome, 'Joao da Silva')
```

Ferramentas extras

Pytest-Django

- integração do framework Pytest com Django para testes mais rápidos e organizados.

coverage.py

- medir a porcentagem de código coberto por testes.