FACULDADE DE TECNOLOGIA DE SÃO JOSÉ DOS CAMPOS
FATEC PROFESSOR JESSEN VIDAL

PEDRO HENRIQUE CERQUEIRA FERNANDES

IMPLEMENTAÇÃO DE TESTES AUTOMATIZADOS NA API DO VANGUARDA REPÓRTER

Orientador: Emanuel Mineda Carneiro

São José dos Campos
2020

SUMÁRIO

1 INTRODUÇÃO

É cada vez mais importante para uma emissora de televisão regional, aproximar o público e mostrar conteúdo direcionado, a afiliada a rede Globo no vale do Paraíba, Vanguarda, possui o aplicativo Vanguarda Repórter, nele é possível o telespectador enviar notícias locais participar de concursos culturais e enquetes da emissora, é uma forma prática da equipe de jornalismo receber informações que podem se tornar notícias importantes nos jornais que ela apresenta. A Vanguarda possui um site interno e com ele a equipe de jornalismo recebe e analisa as notícias enviadas, envia mensagens para os telespectadores e cria os concursos e enquetes.

O aplicativo e o site em questão são atualmente alimentados por uma API escrita em Ruby com o Framework Ruby on Rails, a empresa Necto Systems foi contratada para atualizar o aplicativo, que irá receber novas funcionalidades, alterar a API para Python com o Framework Django, inserir análises de dados com inteligência artificial, que ajudará a equipe de jornalismo nas verificações das notícias enviadas em um novo site interno.

Durante o desenvolvimento de um projeto de software mesmo com planejamento, comportamentos inesperados podem acontecer. Verificar e revisar constantemente as funcionalidades não é só uma forma antiquada de desenvolvimento como improdutivo também. A implantação de testes automatizados é a solução para um desenvolvimento eficiente.

Seguem algumas vantagens do uso de testes no desenvolvimento

- **Tempo**: mesmo que no início pareça que está se gastando mais tempo, mas na verdade se está economizando, mais pra frente com a medida que o projeto fica mais complexo.
- **Custo**: segundo o livro Engenharia de Software de Pressman o custo de implementar o teste no projeto pode ser 100 vezes menor comparado aos ter que encontrar e reparar falhas.
- **Qualidade**: para garantir a qualidade de qualquer produto testes são necessários isso desde sistemas simples até os mais complexos.

1.1 Definição do problema

Conforme o projeto aumentava de tamanho e complexibilidade, sua manutenção e continuidade foi se tornando mais lenta, já que passava a surgir cada vez mais retrabalho por métodos que vinham a entrar em conflitos com outros, já implementados, fazendo com que sempre fosse preciso dar um passo para trás.

1.2 Objetivo

A motivação deste trabalho é garantir a qualidade e diminuir futuros gastos e retrabalhos por meio da implementação de testes de software.

2 DESENVOLVIMENTO

2.1 Arquitetura

A arquitetura geral do sistema pode ser observada na imagem a seguir, sendo que:

- **Cliente** – usuários que acessam as aplicações:
  - **Aplicação WEB** – Vanguarda Admin.
  - **Mobile Nativo** – Aplicativo Vanguarda Repórter.
- **Comunicação** – Conectivo entre o Cliente e o Armazenamento, onde ocorre o controle de rotas e o processamento das regras de negócio.
  - **NGINX Proxy Reverso** – Mediador entre aplicações e servidores.
  - **Vue NGINX –** Lugar em que é feito o build dos arquivos estáticos.
  - **API Aplicação WEB/Mobile** – API REST respo
- **Armazenamento / Persistência** – Ambientes em que são armazenados os dados do sistema, banco de dados.
  - **PostgreSQL** – Banco de dados.
  - **AWS** – Responsável por armazenar as mídias enviadas pelo aplicativo mobile.

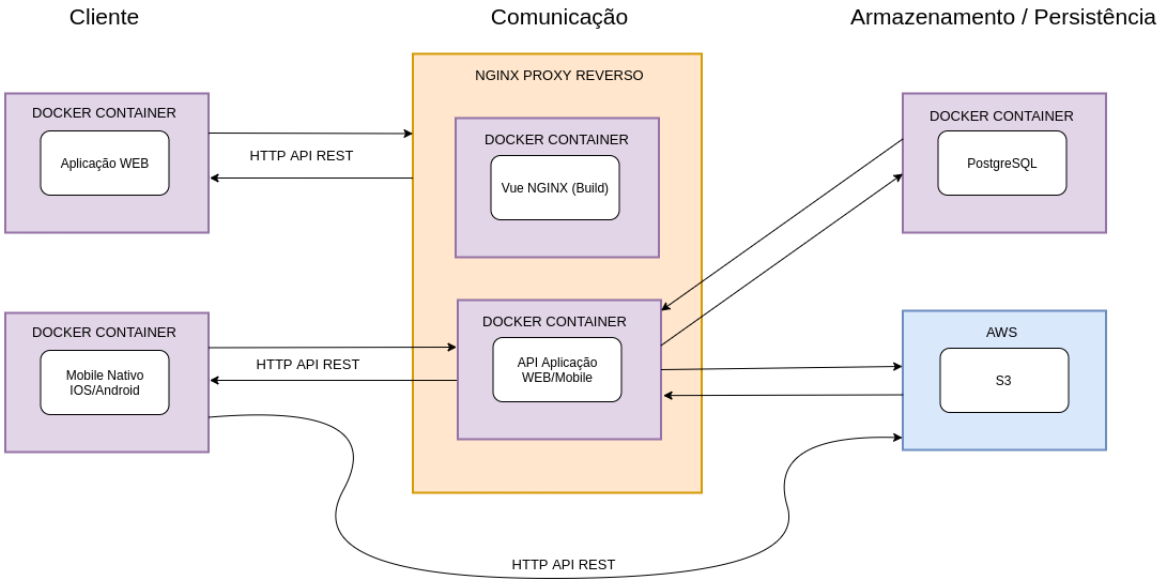


Diagrama 1 – Arquitetura do sistema

2.2 Modelo de Dados

A próxima figura apresenta o modelo de dados do sistema, uma representação ilustrativa de como os dados interagem dentro do banco de dados PostgreSQL, contendo as seguintes tabelas:

- **mobile_registers** – forma antiga de armazenamento dos usuários cadastrados no aplicativo (versões anteriores).
- **users** – usuários cadastrados no aplicativo.
- **feeds** – possíveis notícias enviadas pelos usuários do aplicativo VR.
- **assets** – hashs das notícias enviadas para verificação no repositório da AWS..
- **comments** – comentários dos administradores do site interno nas notícias exibidas.
- **feeds_shows** – exibição das notícias e os respectivos pontos.
- **programs** – programas da grade de notícias da vanguarda.
- **admins** – usuários do sistema site interno(funcionários da Vanguarda).
- **groups** – grupos de usuários do site interno.
- **access_rules** – regras de acesso dos usuários do site interno.
- **access_rules_groups** – regras de acesso por grupo do site interno.
- **logs** – log dos usuários do site interno.
- **competitions** – concursos criados para os usuários do aplicativo participarem.

- **polls** – enquetes para os usuários do aplicativo.
- **poll_alternatives** – alternativas de uma enquete específica.
- **poll_answers** – respostas das enquetes dadas pelos usuários do aplicativo.
- **messages** – mensagens dos usuários do aplicativo enviadas aos usuários do site interno como notificações.
- **tokens** - tokens para acesso via JWT.

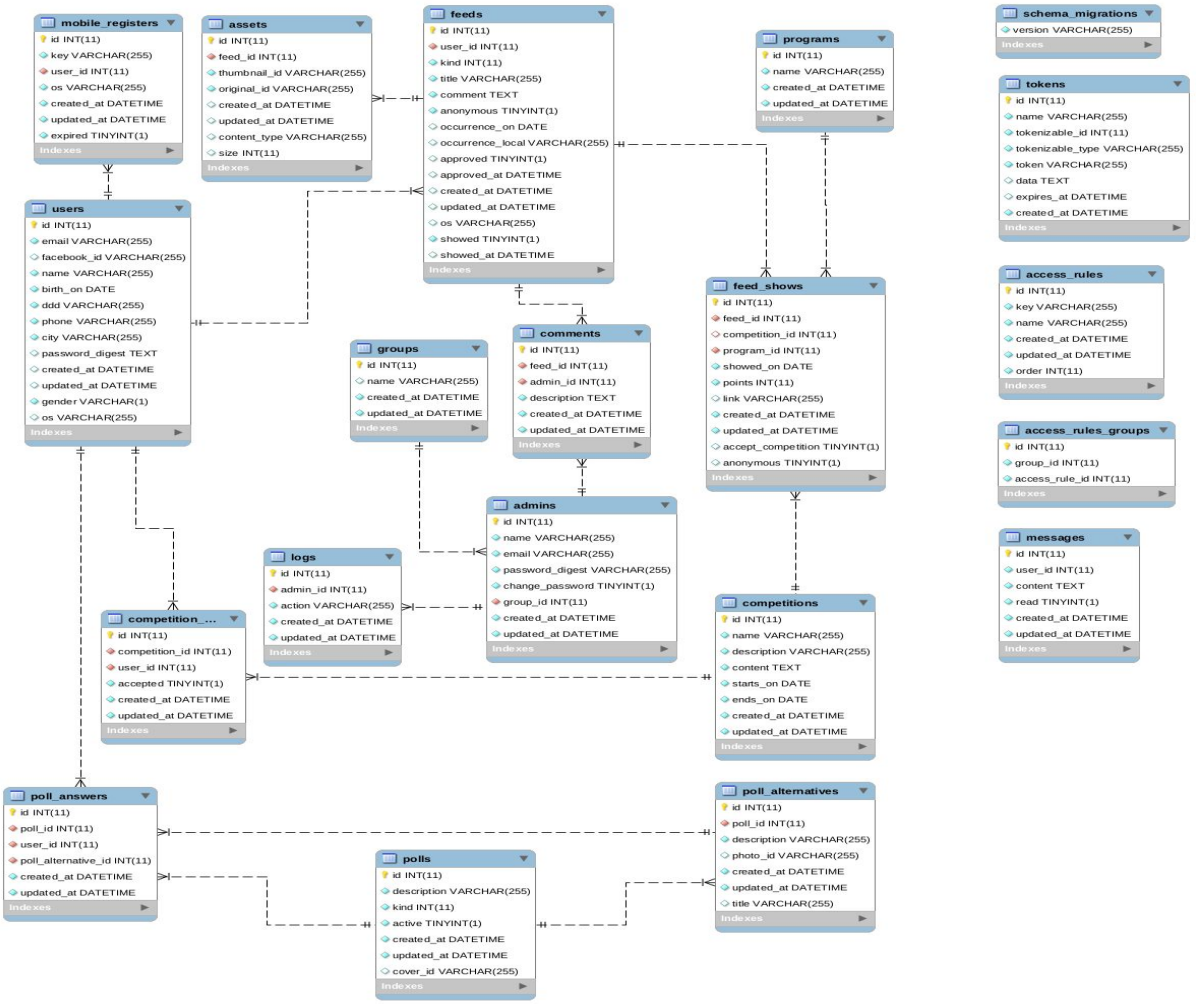


Diagrama 2 – Banco de Dados

2.3 Detalhes

Está sendo utilizado o Coverage para analisar a quantidade de código testado, inicialmente para cobrir as funcionalidades principais da API tanto o que alimenta o aplicativo quanto o site interno, conforme a imagem abaixo até o momento 78% do código atual passa pelos testes.

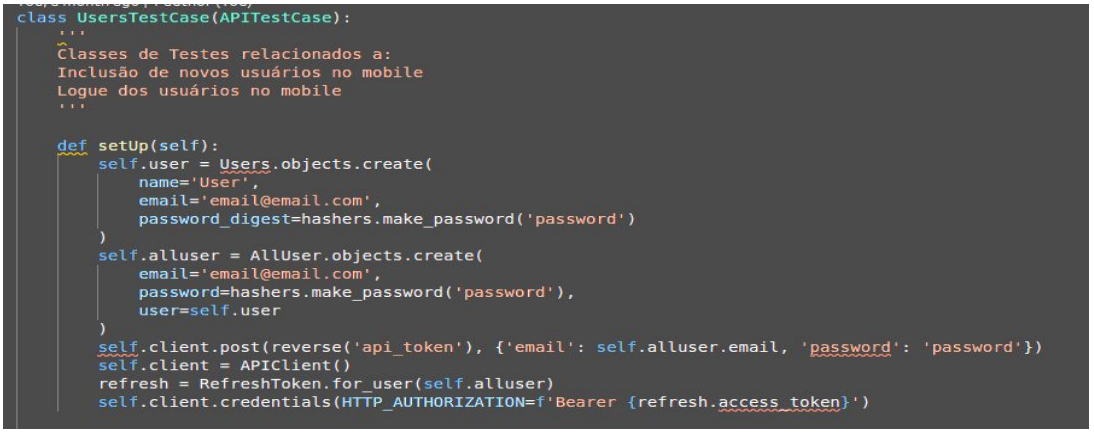
Coverage report: 78%

Module ↑	statements	missing	excluded	coverage
api_vanguarda/__init__.py	0	0	0	100%
api_vanguarda/settings/base.py	37	0	0	100%
api_vanguarda/settings/tests.py	7	0	0	100%
api_vanguarda/urls.py	52	0	0	100%
api_vanguarda/wsgi.py	4	4	0	0%
manage.py	12	2	0	83%
noticias/__init__.py	0	0	0	100%
noticias/admin.py	125	25	0	80%
noticias/apps.py	3	3	0	0%
noticias/autorizacao_antiga.py	20	6	0	70%
noticias/forms.py	30	0	0	100%
noticias/middleware.py	10	10	0	0%
noticias/models.py	416	102	0	75%
noticias/pagination.py	5	0	0	100%
noticias/serializers.py	191	65	0	66%
noticias/test/__init__.py	0	0	0	100%
noticias/test/tests_models.py	26	0	0	100%
noticias/test/tests_views.py	104	0	0	100%
noticias/test/tests_viewsets.py	162	20	0	88%
noticias/views.py	287	126	0	56%
noticias/viewsets.py	249	16	0	94%
Total	**1740**	**379**	**0**	**78%**

Figura 1 – Cobertura Coverage

Como o projeto já estava iniciado, decidimos começar com os testes de saídas de alguns endpoints, focando de forma inicial em testes funcionais.
Seguem alguns exemplos desses testes funcionais:

- Criação de um novo usuário do aplicativo:



```python
class UsersTestCase(APITestCase):
    '''
    Classes de Testes relacionados a:
    Inclusão de novos usuários no mobile
    Logue dos usuários no mobile
    '''

    def setUp(self):
        self.user = Users.objects.create(
            name='User',
            email='email@email.com',
            password_digest=hashers.make_password('password')
        )
        self.alluser = AllUser.objects.create(
            email='email@email.com',
            password=hashers.make_password('password'),
            user=self.user
        )
        self.client.post(reverse('api_token'), {'email': self.alluser.email, 'password': 'password'})
        self.client = APIClient()
        refresh = RefreshToken.for_user(self.alluser)
        self.client.credentials(HTTP_AUTHORIZATION=f'Bearer {refresh.access_token}')
```

Figura 2 – Setup classe de teste do usuário do aplicativo

```python
def test_create_user(self):
    self.client.credentials(HTTP_AUTHORIZATION=settings.TOKEN_FIXO_VANGUARDA)
    data = {
        "user":
        {
            "facebook_id": "",
            "name": "Nome Usuário1231",
            "email": "novousuarioteste0010@email.com.br",
            "password": "teste",
            "birth_on": "2000-01-01",
            "ddd": "012",
            "phone": "555-2565",
            "city": "S. J. dos Campos",
            "os": "android"
        }
    }
    url = reverse('api_users')
    response = self.client.post(url, data, format='json')
    self.assertEqual(response.status_code, status.HTTP_200_OK)
```

Figura 3 – Teste de criação de um novo usuário do aplicativo

```python
def test_create_user_fail(self):
    data = {
        "user":
        {
            "facebook_id": "",
            "name": "Nome Usuário1231",
            "email": "novousuarioteste0010@email.com.br",
            "password": "teste",
            "birth_on": "2000-01-01",
            "ddd": "012",
            "phone": "555-2565",
            "city": "S. J. dos Campos",
            "os": "android"
        }
    }
    url = reverse('mobile_users')
    # criação de um novo usuario
    response = self.client.post(url, data, format='json')
    # tentativa de criar um usuário ja existente resposta esperada status HTTP 400_BAD_REQUEST
    response_fail = self.client.post(url, data, format='json')
    self.assertEqual(response_fail.status_code, status.HTTP_400_BAD_REQUEST)
```

Figura 4 – Teste de criação de usuário do aplicativo já existente no banco

- Criação e visualização de um novo usuário do site interno:

```
class AdminTestCase(APITestCase):
    '''
    Classes de Testes relacionados a:
    Inclusão de novos funcionarios---
    Visualizar todos os funcionarios
    Visualizar dados de um funcionario específico
    Alterar dados de um funcionário
    Deletar um funcionário
    '''

    def setUp(self):
        self.group = Groups.objects.create(
        )
        self.admin_ativo = Admins.objects.create(
            name='Admin',
            email='email@email.com',
            password_digest=hashers.make_password('password'),
            group=self.group
        )
        self.admin_inativo = Admins.objects.create(
            name='Admin',
            email='email@email_inativo.com',
            password_digest=hashers.make_password('password'),
            is_active=False,
            group=self.group
        )
        self.alluser_ativo = AllUser.objects.create(
            email='email@email_ativo.com',
            password=hashers.make_password('password'),
            admin=self.admin_ativo
        )
        self.alluser_inativo = AllUser.objects.create(
            email='email@email_inativo.com',
            password=hashers.make_password('password'),
            admin=self.admin_inativo
        )
        self.client.post("/api/token/", {'email': self.alluser_ativo.email, 'password': 'password'})
        self.client = APIClient()
        refresh = RefreshToken.for_user(self.alluser_ativo)
        self.client.credentials(HTTP_AUTHORIZATION=f'Bearer {refresh.access_token}')
```

Figura 5 – Setup classe de teste do usuário do aplicativo

```
    def test_admin_create(self):
        data = {
            'name': 'novo_tesste',
            'email': 'novo_tesste@teste1234.com',
            'password': hashers.make_password('password'),
            'group': self.group.pk
        }
        url = reverse('admins_front-list')
        response = self.client.post(url, data,)
        self.assertEqual(response.status_code, status.HTTP_200_OK)
```

Figura 6 – Teste de criação de usuário do site interno.

```python
def test_admin_create_fail(self):
    data = {
        'name': 'novo_tesste',
        'email': 'novo_tesste@teste123.com',
        'password': hashers.make_password('password'),
        'group': self.group.pk
    }
    url = reverse('admins_front-list')
    # criação de um novo usuario
    response = self.client.post(url, data)
    # tentativa de criar um usuário ja existente resposta esperada status HTTP 400_BAD_REQUEST
    response_fail = self.client.post(url, data)
    self.assertEqual(response_fail.status_code, status.HTTP_400_BAD_REQUEST)
```

Figura 7 – Teste de criação de usuário do site interno já existente no banco

```python
def test_admin_get(self):
    url = reverse('admins_front-detail', args=[self.admin_ativo.pk])
    response = self.client.get(url)
    self.assertEqual(response.status_code, status.HTTP_200_OK)
```

Figura 8 – Teste de visualização de usuário do site interno ativo.

```python
def test_admin_get_fail(self):
    url = reverse('admins_front-detail', args=[self.admin_inativo.pk])
    response = self.client.get(url)
    self.assertEqual(response.status_code, status.HTTP_404_NOT_FOUND)
```

Figura 9 – Teste de visualização de usuário do site interno inativo.

3 Resultados e Discussão

O desenvolvimento da API apresentou grande melhora devido a implantação dos testes tanto na qualidade do código, agora orientado a testes, quanto na velocidade de desenvolvimento, já que agora a cada nova funcionalidade desenvolvida testes verificam se as funcionalidades já existentes não deixam de funcionar, com isso o custo e o tempo final do projeto serão menores, satisfazendo tanto o cliente final quanto o time de desenvolvimento do mobile e o time de front-end do site interno da empresa.