```
from django.urls import reverse
from rest_framework import status
from rest_framework.test import APITestCase
from django.contrib.auth import get_user_model
from rest_framework_simplejwt.tokens import RefreshToken
from unittest.mock import Mock, patch
from users.models import UserProfile
from rest_framework.response import Response

User = get_user_model()

class UserGoogleLoginTests(APITestCase):
    @patch('users.views.GoogleLoginView.get_google_user_info')
    @patch('dj_rest_auth.registration.views.SocialLoginView.post')
    def test_google_login_new_user(self, mock_super_post,
mock_get_google_user_info):
        # Mock the response from the Google API
        mock_get_google_user_info.return_value = {
            'email': 'testuser@example.com',
            'given_name': 'Test',
            'family_name': 'User'
        }

        # Simulate the response from super().post()
        mock_super_post.return_value = Response(status=status.HTTP_200_OK)

        # Now make a request to the Google login endpoint
        url = reverse('auth_social_google')
        data = {'access_token': 'fake-access-token'}
        response = self.client.post(url, data, format='json')

        # Assert that the response is successful
        self.assertEqual(response.status_code, status.HTTP_200_OK)

        # Check that the user was created
        user = User.objects.get(email='testuser@example.com')
        self.assertIsNotNone(user)
        self.assertEqual(user.first_name, 'Test')
        self.assertEqual(user.last_name, 'User')
        # Check that tokens are returned in the response
        self.assertIn('access', response.data)
        self.assertIn('refresh', response.data)

    @patch('users.views.GoogleLoginView.get_google_user_info')
    @patch('dj_rest_auth.registration.views.SocialLoginView.post')
    def test_google_login_existing_user(self, mock_super_post,
mock_get_google_user_info):
        # Create a user that already exists
        existing_user = User.objects.create_user(
            username='testuser',
            email='testuser@example.com',
            first_name='Existing',
            last_name='User'
        )

        # Mock the response from the Google API
        mock_get_google_user_info.return_value = {
            'email': 'testuser@example.com',
            'given_name': 'Existing',
            'family_name': 'User'
        }

        # Simulate the response from super().post()
        mock_super_post.return_value = Response(status=status.HTTP_200_OK)

        # Now make a request to the Google login endpoint
        url = reverse('auth_social_google')
        data = {'access_token': 'fake-access-token'}
        response = self.client.post(url, data, format='json')

        # Assert that the response is successful
        self.assertEqual(response.status_code, status.HTTP_200_OK)

        # Check that the user still exists and was not duplicated
        user_count =
User.objects.filter(email='testuser@example.com').count()
        self.assertEqual(user_count, 1)

        # Check that tokens are returned in the response
        self.assertIn('access', response.data)
        self.assertIn('refresh', response.data)


class UserProfileTests(APITestCase):
    def setUp(self):
        # Create a test user and retrieve tokens
        self.user = User.objects.create_user(
            username='testuser',
            email='testuser@example.com',
            password='testpassword',
            first_name='Test',
            last_name='User'
        )
        self.refresh_token = RefreshToken.for_user(self.user)
        self.access_token = str(self.refresh_token.access_token)

    def test_get_user_profile(self):
        # Authorize with JWT token
        self.client.credentials(HTTP_AUTHORIZATION='Bearer ' +
self.access_token)
        url = reverse('user_profile')
        response = self.client.get(url)

        # Assert profile data is returned successfully
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertIn('email', response.data)
        self.assertEqual(response.data['email'], self.user.email)

    def test_update_user_profile(self):
        self.client.credentials(HTTP_AUTHORIZATION='Bearer ' +
self.access_token)
        url = reverse('user_profile')
        data = {
            'first_name': 'Updated',
            'last_name': 'User'
        }
        response = self.client.put(url, data, format='json')

        # Assert profile update is successful
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertEqual(response.data['first_name'], 'Updated')
        self.assertEqual(response.data['last_name'], 'User')

    def test_delete_user_profile(self):
        self.client.credentials(HTTP_AUTHORIZATION='Bearer ' +
self.access_token)
        url = reverse('user_profile')
        response = self.client.delete(url)

        # Assert deletion is successful
        self.assertEqual(response.status_code, status.HTTP_204_NO_CONTENT)
        self.assertFalse(User.objects.filter(id=self.user.id).exists())


class UserLogoutTests(APITestCase):
    def setUp(self):
        # Create a test user and retrieve tokens
        self.user = User.objects.create_user(
            username='testuser',
            email='testuser@example.com',
            password='testpassword'
        )
        self.refresh_token = RefreshToken.for_user(self.user)
        self.access_token = str(self.refresh_token.access_token)

    def test_logout(self):
        self.client.credentials(HTTP_AUTHORIZATION='Bearer ' +
self.access_token)
        url = reverse('auth_logout')
        data = {'refresh': str(self.refresh_token)}
        response = self.client.post(url, data, format='json')

        # Assert logout is successful and token is blacklisted
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertIn('Successfully logged out', response.data['detail'])
```