backend\api\tests.py

```python
from django.test import TestCase
from django.urls import reverse
from rest_framework import status
from rest_framework.test import APIClient
from rest_framework.authtoken.models import Token
from .models import User, PreMedicalForm, BlogPost, Comment
from .serializers import UserSerializer, CommentSerializer
from .decorators import role_required
from django.http import HttpResponseForbidden

class APITests(TestCase):
    def setUp(self):
        self.client = APIClient()
        self.user = User.objects.create_user(username='testuser', password='testpassword', role='patient')
        self.token = Token.objects.create(user=self.user)
        self.pre_medical_form = PreMedicalForm.objects.create(patient=self.user, symptoms='Test symptoms')
        self.blog_post = BlogPost.objects.create(title='Test Blog Post', author=self.user, content='Test content', pre_medical_form=self.pre_medical_form)
        self.comment = Comment.objects.create(content='Test Comment', author=self.user, blog_post=self.blog_post, commenter_role='patient')

    def test_index(self):
        url = reverse('index')
        self.client.credentials(HTTP_AUTHORIZATION=f'Token {self.token.key}')
        response = self.client.get(url)
        self.assertEqual(response.status_code, status.HTTP_200_OK)

    def test_register(self):
        url = reverse('register')
        data = {'username': 'newuser', 'password': 'newpassword', 'role': 'patient'}
        response = self.client.post(url, data)
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)

    def test_user_delete_view(self):
        url = reverse('delete-user', kwargs={'pk': self.user.id})
        response = self.client.delete(url)
        self.assertEqual(response.status_code, status.HTTP_204_NO_CONTENT)
        self.assertFalse(User.objects.filter(pk=self.user.id).exists())

    def test_delete_view(self):
        model = Comment
        obj = self.comment

        url = reverse(f'delete-{model.__name__.lower()}', kwargs={'pk': obj.id})
        response = self.client.delete(url)
        self.assertEqual(response.status_code, status.HTTP_204_NO_CONTENT)
        self.assertFalse(model.objects.filter(pk=obj.id).exists())

    def test_comment_delete_view(self):
        self.test_delete_view()

    def test_login_view(self):
        url = reverse('login_view')
        data = {'username': 'testuser', 'password': 'testpassword'}
        response = self.client.post(url, data)
        self.assertEqual(response.status_code, status.HTTP_200_OK)

    def test_create_blog_post(self):
        url = reverse('create_blog_post')
        data = {'title': 'New Blog Post', 'content': 'Test content'}
        self.client.credentials(HTTP_AUTHORIZATION=f'Token {self.token.key}')
        response = self.client.post(url, data, format='json')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)

    def test_create_comment(self):
        url = reverse('create_comment')
        data = {'content': 'New Comment', 'blog_post': self.blog_post.id}
        self.client.credentials(HTTP_AUTHORIZATION=f'Token {self.token.key}')
        response = self.client.post(url, data, format='json')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)

    def test_user_details(self):
        url = reverse('user_details')
        self.client.credentials(HTTP_AUTHORIZATION=f'Token {self.token.key}')
        response = self.client.get(url)
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertEqual(response.data['username'], 'testuser')

class SerializersTests(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='testuser', password='testpassword', role='patient')
        self.pre_medical_form = PreMedicalForm.objects.create(patient=self.user, symptoms='Test symptoms')
        self.blog_post = BlogPost.objects.create(title='Test Blog Post', author=self.user, content='Test content', pre_medical_form=self.pre_medical_form)
        self.comment = Comment.objects.create(content='Test Comment', author=self.user, blog_post=self.blog_post, commenter_role='patient')

    def test_user_serializer(self):
        serializer = UserSerializer(instance=self.user)
        self.assertEqual(serializer.data['username'], 'testuser')

    def test_comment_serializer(self):
        serializer = CommentSerializer(instance=self.comment)
        self.assertEqual(serializer.data['content'], 'Test Comment')


class DecoratorsTests(TestCase):
    def test_role_required_decorator(self):
        @role_required(allowed_roles=['admin'])
        def sample_view(request):
            return HttpResponseForbidden("Forbidden")

        # Test when the user has the allowed role
        user = User.objects.create_user(username='alloweduser', password='testpassword', role='admin')
        request = self.client.get('/')
        request.user = user
        response = sample_view(request)
        self.assertEqual(response.status_code, 403)

        # Test when the user doesn't have the allowed role
        user = User.objects.create_user(username='nonalloweduser', password='testpassword', role='patient')
        request = self.client.get('/')
        request.user = user
        response = sample_view(request)
        self.assertEqual(response.status_code, 403)

class ModelsTests(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='testuser', password='testpassword', role='patient')
        self.pre_medical_form = PreMedicalForm.objects.create(patient=self.user, symptoms='Test symptoms')
        self.blog_post = BlogPost.objects.create(title='Test Blog Post', author=self.user, content='Test content', pre_medical_form=self.pre_medical_form)
        self.comment = Comment.objects.create(content='Test Comment', author=self.user, blog_post=self.blog_post, commenter_role='patient')

    def test_user_model(self):
        self.assertEqual(self.user.username, 'testuser')
        self.assertEqual(self.user.role, 'patient')

    def test_comment_model(self):
        self.assertEqual(self.comment.content, 'Test Comment')



```