

Teste Permissões Campo Modelo

User

I have these Django classes:

```
class User(PermissionsMixin, AbstractBaseUser, AbstractBaseModel):
    """
    Table contains cognito-users & django-users.

    PermissionsMixin leverages the built-in django model permissions system
    (which allows to limit information for staff users via Groups).
    Note: Django-admin user and app user are not split in different tables because of simplicity of development.
    Some libraries assume there is only one user model, and they can't work with both.
    For example, to have a history log of changes for entities - to save which
    user made a change of object attribute, perhaps, auth-related libs, and some
    other.
    With current implementation, we don't need to fork, adapt and maintain third party packages.
    They should work out of the box.
    The disadvantage is - cognito-users will have unused fields which always empty. Not critical.
    """
    username_validator = UnicodeUsernameValidator()

    # Common fields #
    # For cognito-users username will contain 'sub' claim from jwt token
    # (unique identifier (UUID) for the authenticated user).
    # For django-users it will contain username which will be used to login into django-admin site
    username = models.CharField(
        unique=True, max_length=255, unique=True, validators=[username_validator]
    )
    is_active = models.BooleanField("Active", default=True)

    # Cognito-user related fields #
    # some additional fields which will be filled-out only for users registered via Cognito
    pass

    # Django-user related fields #
    # password is inherited from AbstractBaseUser
    email = models.EmailField("Email address", blank=True) # allow non-unique emails
    is_staff = models.BooleanField(
        "staff status",
        default=False,
        help_text="Designates whether the user can log into this admin site.",
    )

    first_name = models.CharField(max_length=255, blank=True)
    last_name = models.CharField(max_length=255, blank=True)
    gmail = models.EmailField(blank=True)
    preferred_email = models.EmailField(blank=True)

    # user status = models.ForeignKey(user_status_type, on_delete=models.PROTECT)
    # current_practice_area = models.ManyToManyField("PracticeArea")
    # target_practice_area = models.ManyToManyField("PracticeArea")

    current_job_title = models.CharField(max_length=255, blank=True)
    target_job_title = models.CharField(max_length=255, blank=True)
    current_skills = models.CharField(max_length=255, blank=True)
    target_skills = models.CharField(max_length=255, blank=True)

    # desired_roles = models.ManyToManyField("Role")
    # availability = models.IntegerField() # not in ERD, is a separate table. Want to confirm to remove this
    # referred_by = models.ForeignKey(referrer, on_delete=models.PROTECT) # FK to referrer

    linkedin_account = models.CharField(max_length=255, blank=True)
    github_handle = models.CharField(max_length=255, blank=True)
    slack_id = models.CharField(max_length=11, blank=True)

    phone = PhoneNumberField(blank=True)

    texting_ok = models.BooleanField(default=True)

    time_zone = TimeZoneField(blank=True, use_pytz=False, default="America/Los_Angeles")
    # conduct = models.BooleanField() # not in ERD. Maybe we should remove this

    objects = UserManager()

    USERNAME_FIELD = "username"
    EMAIL_FIELD = "preferred_email"
    REQUIRED_FIELDS = ["email"] # used only on createsuperuser

    @property
    def is_django_user(self):
        return getattr(self, "password", None) is not None

    def __str__(self):
        return f"{self.username}"

class Project(AbstractBaseModel):
    """
    List of projects
    """
    name = models.CharField(max_length=255, unique=True)
    description = models.CharField(max_length=255, blank=True)
    completion_date = models.DateTimeField("Completed at", null=True, blank=True)
    github_org_id = models.CharField(
        max_length=8,
        blank=True,
        help_text="Can be retrieved from gh api with the following: curl -H \
        'Authorization: token [gh_PAT]' https://api.github.com/orgs/[org]",
    )
    github_primary_repo_id = models.CharField(
        max_length=9,
        blank=True,
        help_text="Can be retrieved from gh api with the following: curl -H \
        'Authorization: token [gh_PAT]' \
        https://api.github.com/repos/[org]/[repo]",
    )
    # current_status_id = models.ForeignKey("status", on_delete=models.PROTECT)
    hide = models.BooleanField(default=True)
    # location_id = models.ForeignKey("location", on_delete=models.PROTECT)
    google_drive_id = models.CharField(max_length=255, blank=True)
    # leads = models.ManyToManyField("lead")
    # leadership_type_id = models.ForeignKey("leadership_type", on_delete=models.PROTECT)
    image_logo = models.URLField(blank=True)
    image_hero = models.URLField(blank=True)
    image_icon = models.URLField(blank=True)

    def __str__(self):
        return f"{self.name}"

class PracticeArea(AbstractBaseModel):
    """
    Practice Area
    """
    name = models.CharField(max_length=255, unique=True)
    description = models.CharField(max_length=255, blank=True)

    def __str__(self):
        return f"{self.name}"

class PermissionType(AbstractBaseModel):
    """
    Permission Type
    """
    name = models.CharField(max_length=255)
    description = models.TextField(blank=True)

    def __str__(self):
        if self.description and isinstance(self.description, str):
            return f"{self.name}: {self.description}"
        else:
            return f"{self.name}"

class Permission(AbstractBaseModel):
    user_id = models.ManyToManyField(User, unique=True)
    project_id = models.ManyToManyField(Project, unique=True)
    practice_area_id = models.ManyToManyField(PracticeArea, unique=True)
    permission_type_id = models.ManyToManyField(PermissionType, unique=True)
    granted = models.DateField(blank=True)
    ended = models.DateField(blank=True)

    def __str__(self):
        return f"{self.user_id}"

I want to test field-level permissions on these classes. Suggest me a way to do that.
```

ChatGPT

To test field-level permissions in Django, you can utilize the Django testing framework along with the 'django.contrib.auth' module. Here's a basic guide on how you can approach testing field-level permissions for your Django models:

- Setup Test Environment:** Ensure that you have configured your Django test environment properly. This typically involves creating test databases and configuring settings for your tests.
- Create Test Cases:** Write test cases to simulate different scenarios where field-level permissions need to be tested. This involves creating users with different permissions and roles and then attempting to access or modify fields based on those permissions.
- Use Django's TestCase Class:** Django provides a 'TestCase' class which provides various helper methods to facilitate testing. You can subclass this to create your test cases.
- Test Field Access:** Write test methods to check whether users with specific permissions can access or modify fields as expected. For example, you can check if a user with staff privileges can modify certain fields while a regular user cannot.
- Use Django's Permission System:** Django's built-in permission system allows you to define custom permissions for models. You can assign these permissions to users and groups and then check if users with those permissions can perform certain actions.
- Test CRUD Operations:** Ensure that your test cases cover all CRUD (Create, Read, Update, Delete) operations on your models and that permissions are enforced correctly for each operation.

Here's a simplified example of how you might write such test cases:

```
"""python
from django.test import TestCase
from django.contrib.auth.models import User
from myapp.models import Project

class FieldPermissionsTestCase(TestCase):
    def setUp(self):
        self.user = User.objects.create(username='test_user', is_staff=True)
        self.project = Project.objects.create(name='Test Project')

    def test_field_access(self):
        # Check if a staff user can modify certain fields
        self.assertTrue(self.user.is_staff)
        self.user.first_name = 'John'
        self.user.save()
        self.assertEqual(self.user.first_name, 'John')

        # Check if a regular user cannot modify certain fields
        regular_user = User.objects.create(username='regular_user', is_staff=False)
        regular_user.first_name = 'Jane'
        regular_user.save()
        self.assertNotEqual(regular_user.first_name, 'Jane')

    def test_permission_system(self):
        # Check if a user with specific permission can access certain fields
        # Here, you can create custom permissions and assign them to users or groups
        # Then, test if users with those permissions can access specific fields
        pass

... # Additional test methods for CRUD operations...

Remember to run your tests using Django's test runner ('./manage.py test') to ensure they are executed against your test database. Also, make sure to test edge cases and handle exceptions gracefully in your test cases.
"""
```

User

I would like to know how to test the classes using pytest, as that is what my organization has already been using for tests.

I have already implemented the test data in a Python file:

```
@pytest.fixture
def field_level_user():
    return [
        User.objects.create(
            uuid=1,
            name="Admin",
            phone="555-222-3333",
            texting_ok=True,
            preferred_email="admin@something908.com"
        ),
        User.objects.create(
            uuid=2,
            name="Sarah",
            phone="555-235-8989",
            texting_ok=True,
            preferred_email="sarah@something908.com"
        ),
        User.objects.create(
            uuid=3,
            name="Bob",
            phone="555-456-7890",
            texting_ok=False,
            preferred_email="bob@something908.com"
        ),
        User.objects.create(
            uuid=4,
            name="Alice",
            phone="555-765-4321",
            texting_ok=True,
            preferred_email="alice@something908.com"
        ),
        User.objects.create(
            uuid=5,
            name="Joe",
            phone="555-468-5656",
            texting_ok=False,
            preferred_email="joe@something908.com"
        ),
        User.objects.create(
            uuid=6,
            name="snoop",
            phone="555-555-5656",
            texting_ok=False,
            preferred_email="snoop@something908.com"
        ),
        User.objects.create(
            uuid=7,
            name="Ralph",
            phone="555-555-8888",
            texting_ok=True,
            preferred_email="ralph@something908.com"
        ),
        User.objects.create(
            uuid=8,
            name="Claire",
            phone="555-555-6666",
            texting_ok=True,
            preferred_email="claire@something908.com"
        ),
        User.objects.create(
            uuid=9,
            name="Mary",
            phone="555-555-2222",
            texting_ok=False,
            preferred_email="mary@something908.com"
        ),
    ]

@pytest.fixture
def field_level_permission():
    return [
        Permission.objects.create(
            user_id=1,
            project_id=1,
            practice_area_id=1,
            permission_type_id=1,
            granted="2023-12-01"
        ),
        Permission.objects.create(
            user_id=2,
            project_id=1,
            practice_area_id=2,
            permission_type_id=2,
            granted="2023-12-01"
        ),
        Permission.objects.create(
            user_id=2,
            project_id=2,
            practice_area_id=2,
            permission_type_id=2,
            granted="2024-01-01"
        ),
        Permission.objects.create(
            user_id=3,
            project_id=1,
            practice_area_id=3,
            permission_type_id=3,
            granted="2023-12-01"
        ),
        Permission.objects.create(
            user_id=4,
            project_id=1,
            practice_area_id=3,
            permission_type_id=3,
            granted="2023-12-30"
        ),
        Permission.objects.create(
            user_id=4,
            project_id=1,
            practice_area_id=3,
            permission_type_id=3,
            granted="2023-12-30"
        ),
        Permission.objects.create(
            user_id=4,
            project_id=1,
            practice_area_id=3,
            permission_type_id=4,
            granted="2023-12-01",
            ended="2023-12-30"
        ),
        Permission.objects.create(
            user_id=4,
            project_id=1,
            practice_area_id=3,
            permission_type_id=5,
            granted="2023-12-01",
            ended="2023-12-01"
        ),
        Permission.objects.create(
            user_id=5,
            project_id=1,
            practice_area_id=3,
            permission_type_id=4,
            granted="2023-12-01",
        ),
        Permission.objects.create(
            user_id=5,
            project_id=1,
            practice_area_id=3,
            permission_type_id=5,
            granted="2023-12-01",
        ),
        Permission.objects.create(
            user_id=7,
            project_id=2,
            practice_area_id=3,
            permission_type_id=4,
            granted="2023-12-01",
        ),
        Permission.objects.create(
            user_id=8,
            project_id=1,
            practice_area_id=3,
            permission_type_id=3,
            granted="2023-12-01",
        ),
        Permission.objects.create(
            user_id=9,
            project_id=1,
            practice_area_id=4,
            permission_type_id=4,
            granted="2023-12-01",
        ),
    ]

@pytest.fixture
def field_level_permission_type():
    return [
        PermissionType.objects.create(
            uuid=1,
            name="adminBrigade"
        ),
        PermissionType.objects.create(
            uuid=2,
            name="adminProject"
        ),
        PermissionType.objects.create(
            uuid=3,
            name="practiceLeadProject"
        ),
        PermissionType.objects.create(
            uuid=4,
            name="memberProject"
        ),
        PermissionType.objects.create(
            uuid=5,
            name="memberGeneral"
        )
    ]

@pytest.fixture
def field_level_practice_area():
    return [
        PracticeArea.objects.create(
            uuid=1,
            name="admin"
        ),
        PracticeArea.objects.create(
            uuid=2,
            name="pm"
        ),
        PracticeArea.objects.create(
            uuid=3,
            name="research"
        ),
        PracticeArea.objects.create(
            uuid=4,
            name="design"
        )
    ]

@pytest.fixture
def field_level_project():
    return [
        Project.objects.create(
            uuid=1,
            name="website"
        ),
        Project.objects.create(
            uuid=2,
            name="people depot"
        )
    ]
```

How can I test the classes with pytest using this data?

ChatGPT

To test the classes with pytest using the provided test data fixtures, you can write test functions in your pytest test files. You can create separate test functions for each test case, leveraging the test data provided by the fixtures. Here's how you can structure your pytest test file:

```
"""python
# test_permissions.py

import pytest
from myapp.models import User, Permission, PermissionType, PracticeArea, Project

@pytest.mark.django_db
def test_user_field_level_user():
    users = field_level_user
    assert len(users) == 9
    # Write assertions to test user fields here
    # Example:
    assert users[0].name_first == "Admin"
    assert users[1].name_first == "Sarah"
    # Add more assertions for other user fields

@pytest.mark.django_db
def test_permission_fields(field_level_permission):
    permissions = field_level_permission
    assert len(permissions) == 12
    # Write assertions to test permission fields here
    # Example:
    assert permissions[0].user_id == 1
    assert permissions[1].user_id == 2
    # Add more assertions for other permission fields

@pytest.mark.django_db
def test_permission_type_fields(field_level_permission_type):
    permission_types = field_level_permission_type
    assert len(permission_types) == 5
    # Write assertions to test permission type fields here
    # Example:
    assert permission_types[0].name == "adminBrigade"
    assert permission_types[1].name == "adminProject"
    # Add more assertions for other permission type fields

@pytest.mark.django_db
def test_practice_area_fields(field_level_practice_area):
    practice_areas = field_level_practice_area
    assert len(practice_areas) == 4
    # Write assertions to test practice area fields here
    # Example:
    assert practice_areas[0].name == "admin"
    assert practice_areas[1].name == "pm"
    # Add more assertions for other practice area fields

@pytest.mark.django_db
def test_project_fields(field_level_project):
    projects = field_level_project
    assert len(projects) == 2
    # Write assertions to test project fields here
    # Example:
    assert projects[0].name == "website"
    # Add more assertions for other project fields
...
"""
```

In this file, each test function is marked with '@pytest.mark.django_db' decorator to ensure that the tests run within a Django transaction and have access to the database.

You can run these tests using pytest by executing the following command in your terminal:

```
"""bash
pytest path/to/test_permissions.py
```

Replace 'path/to/test_permissions.py' with the actual path to your pytest test file. This will execute all the test functions defined in the file and report any failures or errors encountered during the tests.

User

I would like to specifically test each permission type in the data (adminBrigade, adminProject, practiceLeadProject, memberProject, and memberGeneral). Each permission_type will have their own field-level visibility settings. How can I do that?

ChatGPT

To test each 'PermissionType' and its corresponding field-level visibility settings, you can create separate test functions for each permission type. In these functions, you can assert that the expected fields are visible or not visible for users with permissions of that type. Here's how you can do it:

```
"""python
# test_permission_types.py

import pytest
from myapp.models import User, Permission, PermissionType, PracticeArea, Project

@pytest.mark.django_db
def test_admin_brigade_permissions(field_level_user, field_level_permission_type):
    # Create test data
    users = field_level_user
    permission_type_admin_brigade = PermissionType.objects.get(name="adminBrigade")
    # Assert that adminBrigade permissions allow access to specific fields
    assert permission_type_admin_brigade.name == "adminBrigade"
    # Add assertions for field-level visibility settings for adminBrigade permissions

@pytest.mark.django_db
def test_admin_project_permissions(field_level_user, field_level_permission_type):
    # Create test data
    users = field_level_user
    permission_type_admin_project = PermissionType.objects.get(name="adminProject")
    # Assert that adminProject permissions allow access to specific fields
    assert permission_type_admin_project.name == "adminProject"
    # Add assertions for field-level visibility settings for adminProject permissions

@pytest.mark.django_db
def test_practice_lead_project_permissions(field_level_user, field_level_permission_type):
    # Create test data
    users = field_level_user
    permission_type_practice_lead_project = PermissionType.objects.get(name="practiceLeadProject")
    # Assert that practiceLeadProject permissions allow access to specific fields
    assert permission_type_practice_lead_project.name == "practiceLeadProject"
    # Add assertions for field-level visibility settings for practiceLeadProject permissions

@pytest.mark.django_db
def test_member_project_permissions(field_level_user, field_level_permission_type):
    # Create test data
    users = field_level_user
    permission_type_member_project = PermissionType.objects.get(name="memberProject")
    # Assert that memberProject permissions allow access to specific fields
    # Add assertions for field-level visibility settings for memberProject permissions

@pytest.mark.django_db
def test_member_general_permissions(field_level_user, field_level_permission_type):
    # Create test data
    users = field_level_user
    permission_type_member_general = PermissionType.objects.get(name="memberGeneral")
    # Assert that memberGeneral permissions allow access to specific fields
    assert permission_type_member_general.name == "memberGeneral"
    # Add assertions for field-level visibility settings for memberGeneral permissions
...
"""
```

In each test function, you can retrieve the relevant 'PermissionType' object from the test data fixtures and then assert its name and field-level visibility settings according to your requirements.

Remember to replace the placeholder comments with actual assertions based on your field-level visibility settings for each permission type.

You can run these tests using pytest in the same way as before:

```
"""bash
pytest path/to/test_permission_types.py
```

This will execute all the test functions defined in the file and report any failures or errors encountered during the tests.