Django Exercise: Project and Task Permissions Management System

This exercise will guide students in creating a **Project and Task Management System** with a focus on implementing and enforcing **Django permissions**. The goal is to cover **user-level**, **group-level**, **model-level**, and **object-level permissions**.

Exercise Overview

You will build a system where:

- Users can create, view, and manage **projects** and associated **tasks**.
- **Permissions** will control access to create, update, and delete both projects and tasks.
- Permissions will be enforced at **user-level**, **group-level**, **model-level**, and **object-level**.
- You will also implement views and templates that reflect permissions for each user.

Tasks

1. Project Setup and Models

1. **Set Up a Django Project and App**:
    - Create a new Django project called `permissioned_manager`.
    - Create an app called `projects`.
2. **Define Models**:
    - **Project Model**: Fields for `name`, `description`, `start_date`, and `end_date`.
    - **Task Model**: Fields for `title`, `description`, `completed`, `due_date`, and a foreign key to `Project`.

---

2. Create and Assign Permissions

1. **User-Level and Group-Level Permissions**:
    - Use Django's **user-level permissions** to restrict access to creating, editing, and deleting projects.
    - Create **groups** like `ProjectManagers` and `TaskEditors`:
        - **ProjectManagers**: Can create, update, and delete projects.
        - **TaskEditors**: Can create, update, and delete tasks within a project.
2. **Model-Level Permissions**:
    - Assign model-level permissions for the `Project` and `Task` models (e.g., `add_project`, `change_project`, `delete_project`, `add_task`, etc.).
    - Ensure that the permissions are only available to the appropriate user groups.

Hints:

- Use Django's `User.has_perm()` method to check if a user has a specific permission.
- Use `Group` and `Permission` models to create groups and assign permissions.

---

3. Object-Level Permissions

- Implement **object-level permissions** so that:
  - Only the project creator or members of the `ProjectManagers` group can edit or delete that project.
  - Only users with permission on a specific `Task` object can edit or delete it.

Hint:

- Use `user.has_perm('change_project', project)` to check if a user has permission for a specific object.
- You may use `django-guardian` for simpler handling of object-level permissions if desired.

---

4. Enforce Permissions in Views

1. **Project Views**:
   - **Project List**: Display projects the user has permission to view.
   - **Project Detail**: Only allow users with view permissions to see project details.
   - **Project Create, Update, and Delete**: Allow access based on user and group permissions.
2. **Task Views**:
   - **Task List and Detail**: Display tasks only if the user has permission to view them.
   - **Task Create, Update, and Delete**: Allow access based on the permissions for tasks within a project.

Hint:

- Use `permission_required` decorator or `UserPassesTestMixin` in class-based views.
- For object-level permissions, use Django's `get_object_or_404()` with permission checks.

---

5. Display Permission-Sensitive Actions in Templates

- Modify the templates to only show actions (e.g., edit/delete buttons) if the user has the required permissions.
- Show different options for users based on their group memberships and permissions.

Hint:

- Use `{% if user.has_perm %}` template tags to conditionally display content based on permissions.

---

6. Testing Permissions

1. **User and Group Permissions Tests**:
   - Write tests to verify that only users with appropriate permissions can create, edit, and delete projects and tasks.
2. **Object-Level Permissions Tests**:
   - Write tests to ensure that users without object-level permissions cannot access specific projects or tasks.

Hint:

- Use Django's `client.login()` for testing permissions with various user roles and groups.

Bonus Challenges

1. **Custom Permission for Task Completion**:
   - Implement a custom permission (`can_complete_task`) that allows only specific users to mark tasks as completed.
2. **Admin Interface for Permissions**:
   - Customize the Django admin to include permission assignments directly from the admin interface.

What to Submit

- A functional Django project with the `permissioned_manager` project and `projects` app.
- Defined **permissions** at user, group, model, and object levels.
- Views and templates that adapt based on the user's permissions.
- Tests covering all permissions.

Learning Outcomes

By completing this exercise, students will learn:

- How to set up **user-level** and **group-level permissions** in Django.
- How to use **model-level** and **object-level permissions** to enforce access control.
- How to manage and test permissions across various Django views and templates.