Solution: Advanced Project and Task Management System

Here's a full solution for the **Project and Task Management System** exercise, implementing class-based views (CBVs), custom managers, signals, caching, sessions, context processors, form validation, and testing.

Step 1: Create a Django Project

1. **Create the project**:

   ```
   bash
   Copy code
   django-admin startproject advanced_project_manager
   cd advanced_project_manager
   ```

2. **Create the app**:

   ```
   bash
   Copy code
   python manage.py startapp projects
   ```

3. **Add the app to `INSTALLED_APPS`** in `settings.py`:

   ```
   python
   Copy code
   INSTALLED_APPS = [
       'projects',
       'django.contrib.admin',
       'django.contrib.auth',
       'django.contrib.contenttypes',
       'django.contrib.sessions',
       'django.contrib.messages',
       'django.contrib.staticfiles',
   ]
   ```

Step 2: Define Models with Custom Managers and QuerySets

1. **Define `Project` and `Task` models in `projects/models.py`**:

```
python
Copy code
from django.db import models
from django.utils import timezone

class ActiveProjectManager(models.Manager):
    def get_queryset(self):
        return super().get_queryset().filter(end_date__gte=timezone.now())
```

```
class Project(models.Model):
    name = models.CharField(max_length=100)
    description = models.TextField()
    start_date = models.DateField()
    end_date = models.DateField()

    # Custom manager
    objects = models.Manager()
    active_projects = ActiveProjectManager()

    def __str__(self):
        return self.name

class TaskManager(models.Manager):
    def completed(self):
        return self.filter(completed=True)

    def incomplete(self):
        return self.filter(completed=False)

class Task(models.Model):
    project = models.ForeignKey(Project, on_delete=models.CASCADE,
related_name="tasks")
    title = models.CharField(max_length=100)
    description = models.TextField()
    completed = models.BooleanField(default=False)
    due_date = models.DateField()

    # Custom manager
    objects = TaskManager()

    def __str__(self):
        return self.title
```

2. **Migrate the database**:

   ```
   bash
   Copy code
   python manage.py makemigrations
   python manage.py migrate
   ```

Step 3: Class-Based Views for Projects and Tasks

1. **Project List View**: In `projects/views.py`:

```
python
Copy code
from django.views.generic import ListView
from django.utils import timezone
from .models import Project

class ProjectListView(ListView):
    model = Project
```

```
    template_name = 'projects/project_list.html'
    context_object_name = 'projects'

    def get_queryset(self):
        # Sort projects by start date
        sort_by = self.request.GET.get('sort', 'start_date')
        return Project.objects.order_by(sort_by)
```

2. **Project Create and Update Views**: In `projects/views.py`:

```
python
Copy code
from django.views.generic.edit import CreateView, UpdateView
from django.urls import reverse_lazy
from .models import Project

class ProjectCreateView(CreateView):
    model = Project
    fields = ['name', 'description', 'start_date', 'end_date']
    template_name = 'projects/project_form.html'
    success_url = reverse_lazy('project_list')

class ProjectUpdateView(UpdateView):
    model = Project
    fields = ['name', 'description', 'start_date', 'end_date']
    template_name = 'projects/project_form.html'
    success_url = reverse_lazy('project_list')
```

3. **Task List and Completion View**: In `projects/views.py`:

```
python
Copy code
from django.shortcuts import get_object_or_404, redirect
from django.views.generic import ListView
from django.views import View
from .models import Task, Project

class TaskListView(ListView):
    model = Task
    template_name = 'projects/task_list.html'
    context_object_name = 'tasks'

    def get_queryset(self):
        project_id = self.kwargs.get('project_id')
        project = get_object_or_404(Project, id=project_id)
        sort_by = self.request.GET.get('sort', 'due_date')
        return project.tasks.order_by(sort_by)

class TaskCompletionToggleView(View):
    def post(self, request, pk):
        task = get_object_or_404(Task, pk=pk)
        task.completed = not task.completed
        task.save()
        return redirect('task_list', project_id=task.project.id)
```

Step 4: Implement Signals for Automatic Task Updates

In `projects/signals.py`:

```
python
Copy code
from django.db.models.signals import post_save
from django.dispatch import receiver
from .models import Task, Project

@receiver(post_save, sender=Task)
def update_project_status(sender, instance, **kwargs):
    project = instance.project
    if not project.tasks.filter(completed=False).exists():
        project.status = 'Completed'
        project.save()
```

Register signals in `projects/apps.py`:

```
python
Copy code
from django.apps import AppConfig

class ProjectsConfig(AppConfig):
    name = 'projects'

    def ready(self):
        import projects.signals
```

Step 5: Caching for Performance Optimization

1. **Enable caching** in `settings.py`:

```
python
Copy code
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
        'LOCATION': 'unique-snowflake',
    }
}
```

2. **Cache the project list view**: In `projects/views.py`:

```
python
Copy code
from django.views.decorators.cache import cache_page
from django.utils.decorators import method_decorator

class CachedProjectListView(ProjectListView):
    @method_decorator(cache_page(30))  # Cache for 30 seconds
    def dispatch(self, *args, **kwargs):
```

```
        return super().dispatch(*args, **kwargs)
```

Step 6: Add Session-Based Task Management

In `projects/views.py`:

```
python
Copy code
class TaskListView(ListView):
    model = Task
    template_name = 'projects/task_list.html'
    context_object_name = 'tasks'

    def get_queryset(self):
        project_id = self.kwargs.get('project_id')
        project = get_object_or_404(Project, id=project_id)
        sort_by = self.request.session.get('task_sort', 'due_date')  # Use
session to store sorting
        return project.tasks.order_by(sort_by)

    def post(self, request, *args, **kwargs):
        self.request.session['task_sort'] = request.POST.get('sort')
        return redirect('task_list',
project_id=self.kwargs.get('project_id'))
```

Step 7: Context Processors for Global Access

1. **Create the context processor** in `projects/context_processors.py`:

```
python
Copy code
from .models import Project

def ongoing_projects(request):
    return {'ongoing_projects': Project.active_projects.all()}
```

2. **Add the context processor** in `settings.py`:

```
python
Copy code
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'projects.context_processors.ongoing_projects',  # Add here
            ],
```

```
        },
    },
]
```

Step 8: Django Forms and Validation

1. **Project form validation** in `projects/forms.py`:

```
python
Copy code
from django import forms
from .models import Project

class ProjectForm(forms.ModelForm):
    class Meta:
        model = Project
        fields = ['name', 'description', 'start_date', 'end_date']

    def clean(self):
        cleaned_data = super().clean()
        start_date = cleaned_data.get('start_date')
        end_date = cleaned_data.get('end_date')
        if end_date < start_date:
            raise forms.ValidationError("End date cannot be before start
date.")
        return cleaned_data
```

2. **Use the form** in your views:

```
python
Copy code
class ProjectCreateView(CreateView):
    form_class = ProjectForm
    template_name = 'projects/project_form.html'
    success_url = reverse_lazy('project_list')
```

Step 9: Testing Models, Views, and Signals

1. **Model and signal testing** in `projects/tests.py`:

```
python
Copy code
from django.test import TestCase
from .models import Project, Task
from datetime import date

class ProjectModelTest(TestCase):
    def test_active_projects(self):
        Project.objects.create(name="Test Project", start_date=date.today(),
end_date=date.today())
        active_projects = Project.active_projects.all()
```

```
        self.assertEqual(len(active_projects), 1)

class TaskSignalTest(TestCase):
    def test_project_completion_signal(self):
        project = Project.objects.create(name="Test Project",
start_date=date.today(), end_date=date.today())
        Task.objects.create(project=project, title="Test Task 1",
due_date=date.today(), completed=True)
        Task.objects.create(project=project, title="Test Task 2",
due_date=date.today(), completed=True)
        project.refresh_from_db()
        self.assertEqual(project.status, "Completed")
```

Step 10: URL Configuration

In `projects/urls.py`:

```
python
Copy code
from django.urls import path
from .views import ProjectListView, ProjectCreateView, ProjectUpdateView,
TaskListView, TaskCompletionToggleView

urlpatterns = [
    path('', ProjectListView.as_view(), name='project_list'),
    path('create/', ProjectCreateView.as_view(), name='project_create'),
    path('<int:pk>/update/', ProjectUpdateView.as_view(),
name='project_update'),
    path('<int:project_id>/tasks/', TaskListView.as_view(),
name='task_list'),
    path('task/<int:pk>/toggle/', TaskCompletionToggleView.as_view(),
name='task_toggle'),
]
```

Conclusion

This solution covers all the advanced features outlined in the exercise, including **custom managers**, **class-based views (CBVs)**, **signals**, **caching**, **sessions**, **context processors**, and **form validation**.