Task Management System with Class-Based Views (CBVs) and Generic Views

This exercise will guide students through creating a **Task Management System** using Django. The goal is to gradually introduce **Class-Based Views (CBVs)**, starting from basic CBVs and progressing towards **Generic Views**. Students will also learn how to handle forms using CBVs, implement mixins, and write tests for both models and views.

Exercise Overview:

You will create a **Task Management System** where:

- Users can create, view, update, and delete tasks using **Class-Based Views**.
- The system will start by implementing views using basic CBVs (without using Django's Generic Views).
- After mastering basic CBVs, students will refactor the views using **Django Generic Views**.
- You will write tests for both **models** and **views** to ensure functionality.

Step-by-Step Guide:

Step 1: Project Setup and Configuration

1. **Create a Django project** called `task_manager`:

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

2. **Create an app** called `tasks`:

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

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

   ```
   python
   Copy code
   INSTALLED_APPS = [
       # Other apps...
       'tasks',
   ]
   ```

4. **Run migrations** to set up the database:

   ```
   bash
   Copy code
   ```

```
python manage.py migrate
```

Step 2: Define the Task Model

1. In `tasks/models.py`, create a `Task` model with fields like `title`, `description`, `completed`, and `due_date`:

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

class Task(models.Model):
    title = models.CharField(max_length=100)
    description = models.TextField(blank=True)
    completed = models.BooleanField(default=False)
    due_date = models.DateField()

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

2. **Make and apply migrations** for the `Task` model:

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

Step 3: Create Task Views Using Basic Class-Based Views (CBVs)

Task List View (Basic Class-Based View)

1. In `tasks/views.py`, create a simple **class-based view** for listing tasks:

```
python
Copy code
from django.shortcuts import render
from django.views import View
from .models import Task

class TaskListView(View):
    def get(self, request):
        tasks = Task.objects.all()
        return render(request, 'tasks/task_list.html', {'tasks': tasks})
```

2. **Template for displaying tasks** (`tasks/templates/tasks/task_list.html`):

```
html
Copy code
<h2>Task List</h2>
<ul>
    {% for task in tasks %}
    <li>{{ task.title }} - {{ task.due_date }} - {% if task.completed
%}Completed{% else %}Not Completed{% endif %}</li>
    {% endfor %}
```

```
</ul>
<a href="{% url 'task_create' %}">Create a New Task</a>
```

Task Create View (Basic Class-Based View)

3. In `tasks/forms.py`, create a **ModelForm** for the `Task` model:

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

class TaskForm(forms.ModelForm):
    class Meta:
        model = Task
        fields = ['title', 'description', 'completed', 'due_date']
```

4. In `tasks/views.py`, create a **class-based view** for creating tasks using `GET` and `POST` methods:

```
python
Copy code
from django.shortcuts import redirect
from .forms import TaskForm

class TaskCreateView(View):
    def get(self, request):
        form = TaskForm()
        return render(request, 'tasks/task_form.html', {'form': form})

    def post(self, request):
        form = TaskForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('task_list')
        return render(request, 'tasks/task_form.html', {'form': form})
```

5. **Template for task creation** (`tasks/templates/tasks/task_form.html`):

```
html
Copy code
<h2>Create a New Task</h2>
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Save Task</button>
</form>
```

Step 4: Update and Delete Tasks with Basic Class-Based Views

Task Update View (Basic Class-Based View)

1. In `tasks/views.py`, create a class-based view to handle task updates:

```
python
```

```
Copy code
from django.shortcuts import get_object_or_404

class TaskUpdateView(View):
    def get(self, request, pk):
        task = get_object_or_404(Task, pk=pk)
        form = TaskForm(instance=task)
        return render(request, 'tasks/task_form.html', {'form': form})

    def post(self, request, pk):
        task = get_object_or_404(Task, pk=pk)
        form = TaskForm(request.POST, instance=task)
        if form.is_valid():
            form.save()
            return redirect('task_list')
        return render(request, 'tasks/task_form.html', {'form': form})
```

Task Delete View (Basic Class-Based View)

2. Create a view to handle task deletion:

```
python
Copy code
class TaskDeleteView(View):
    def get(self, request, pk):
        task = get_object_or_404(Task, pk=pk)
        return render(request, 'tasks/task_confirm_delete.html', {'task':
task})

    def post(self, request, pk):
        task = get_object_or_404(Task, pk=pk)
        task.delete()
        return redirect('task_list')
```

3. **Template for task deletion confirmation** (`tasks/templates/tasks/task_confirm_delete.html`):

```
html
Copy code
<h2>Are you sure you want to delete "{{ task.title }}"?</h2>
<form method="post">
    {% csrf_token %}
    <button type="submit">Yes, delete</button>
    <a href="{% url 'task_list' %}">Cancel</a>
</form>
```

Step 5: Refactoring Using Django Generic Views

Refactor Task Views Using Generic Views

1. **Task List View** (using `ListView`):

```
python
Copy code
from django.views.generic import ListView
```

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

2. **Task Create View** (using `CreateView`):

```
python
Copy code
from django.urls import reverse_lazy
from django.views.generic.edit import CreateView

class TaskCreateView(CreateView):
    model = Task
    form_class = TaskForm
    template_name = 'tasks/task_form.html'
    success_url = reverse_lazy('task_list')
```

3. **Task Update View** (using `UpdateView`):

```
python
Copy code
from django.views.generic.edit import UpdateView

class TaskUpdateView(UpdateView):
    model = Task
    form_class = TaskForm
    template_name = 'tasks/task_form.html'
    success_url = reverse_lazy('task_list')
```

4. **Task Delete View** (using `DeleteView`):

```
python
Copy code
from django.views.generic.edit import DeleteView

class TaskDeleteView(DeleteView):
    model = Task
    template_name = 'tasks/task_confirm_delete.html'
    success_url = reverse_lazy('task_list')
```

Step 6: URL Configuration

1. In `tasks/urls.py`, configure the URLs for the task views:

```
python
Copy code
from django.urls import path
from .views import TaskCreateView, TaskListView, TaskUpdateView,
TaskDeleteView

urlpatterns = [
    path('', TaskListView.as_view(), name='task_list'),
    path('create/', TaskCreateView.as_view(), name='task_create'),
    path('<int:pk>/update/', TaskUpdateView.as_view(), name='task_update'),
    path('<int:pk>/delete/', TaskDeleteView.as_view(), name='task_delete'),
```

```
]
```

2. Include the `tasks` app URLs in the main `task_manager/urls.py`:

```
python
Copy code
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('tasks/', include('tasks.urls')),
]
```

Step 7: Testing Models and Views

Model Testing

1. In `tasks/tests.py`, create tests for the `Task` model:

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

class TaskModelTest(TestCase):
    def test_string_representation(self):
        task = Task(title="Test Task")
        self.assertEqual(str(task), task.title)

    def test_task_due_date(self):
        task = Task(title="Task with Due Date", due_date=date.today())
        self.assertEqual(task.due_date, date.today())
```

View Testing

2. Create tests for the views to ensure tasks are properly created, listed, and deleted:

```
python
Copy code
from django.urls import reverse
from django.test import TestCase
from .models import Task
from datetime import date

class TaskViewTests(TestCase):
    def setUp(self):
        self.task = Task.objects.create(title="Test Task",
description="Test Description", due_date=date.today(), completed=False)

    def test_task_list_view(self):
        response = self.client.get(reverse('task_list'))
        self.assertEqual(response.status_code, 200)
        self.assertContains(response, self.task.title)
```

```
    def test_task_create_view(self):
        response = self.client.post(reverse('task_create'), {
            'title': 'New Task',
            'description': 'New Description',
            'due_date': date.today(),
            'completed': False
        })
        self.assertEqual(response.status_code, 302)  # Redirects after
success
        self.assertEqual(Task.objects.count(), 2)
```

Bonus Challenges

1. **Search Tasks**:
    - Add a search bar to the task list to filter tasks by title.
2. **Task Completion Toggle**:
    - Implement a view that toggles the `completed` status of a task.
3. **Pagination**:
    - Add pagination to the task list view using Django's built-in pagination tools.

What Students Will Learn:

- How to implement **Class-Based Views (CBVs)** with `GET` and `POST` methods.
- How to refactor views using **Django's Generic Views** for better code reuse.
- How to use **forms with CBVs** to handle user input.
- How to implement **CRUD operations** with class-based views.
- Writing **tests for models and views** to ensure the application functions correctly.