## Django Classroom Exercise: Building Your First Django App

In this exercise, students will create a simple blog application using Django. The goal is to cover the essential concepts of Django, such as **data models**, **migrations**, **admin panel**, **views**, **templates**, **URL routing**, and **Django template inheritance**.

---

## Part 1: Setting Up the Django Project

### Step 1: Install Django

1. Make sure Django is installed:

   ```
   bash
   Copy code
   pip install django
   ```

2. Create a new Django project:

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

3. Run the development server to ensure everything is set up:

   ```
   bash
   Copy code
   python manage.py runserver
   ```

4. Visit `http://127.0.0.1:8000/` in your browser, and you should see the default Django welcome page.

---

## Part 2: Creating a Blog App

### Step 2: Create the Blog App

1. Inside the project, create a new app called `blog`:

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

2. Add the `blog` app to `INSTALLED_APPS` in the `myblog/settings.py` file:

```
python
Copy code
INSTALLED_APPS = [
    # other apps
    'blog',
]
```

---

## Part 3: Django Data Models

### Step 3: Define Data Models for Blog

1. In `blog/models.py`, create the following models:
    - **Post**: Represents a blog post with a title, content, and publication date.
    - **Author**: Represents the author of the blog post.

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

class Author(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField()

    def __str__(self):
        return self.name

class Post(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    pub_date = models.DateTimeField(auto_now_add=True)
    author = models.ForeignKey(Author, on_delete=models.CASCADE)

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

### Step 4: Data Model Migrations

1. Create and apply migrations:

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

---

## Part 4: Django Admin

### Step 5: Register Models in the Admin Panel

1. In `blog/admin.py`, register the `Post` and `Author` models to the admin interface:

```
python
Copy code
from django.contrib import admin
from .models import Post, Author

admin.site.register(Post)
admin.site.register(Author)
```

2. Create a superuser to access the admin panel:

   ```
   bash
   Copy code
   python manage.py createsuperuser
   ```

3. Start the server and visit the Django admin at `http://127.0.0.1:8000/admin/`. Log in using the superuser credentials and manage blog posts and authors through the interface.

---

## Part 5: URL Routing and Views

### Step 6: Create Views for Blog Posts

1. In `blog/views.py`, create views for listing all blog posts and displaying a single post:

```
python
Copy code
from django.shortcuts import render, get_object_or_404
from .models import Post

def post_list(request):
    posts = Post.objects.all()
    return render(request, 'blog/post_list.html', {'posts': posts})

def post_detail(request, post_id):
    post = get_object_or_404(Post, id=post_id)
    return render(request, 'blog/post_detail.html', {'post': post})
```

### Step 7: Set Up URL Routing

1. Create a new `urls.py` file in the `blog` directory and define routes for the blog post list and individual post details:

```
python
Copy code
from django.urls import path
from . import views

urlpatterns = [
    path('', views.post_list, name='post_list'),
    path('post/<int:post_id>/', views.post_detail, name='post_detail'),
]
```

2. Include the `blog` URLs in the main project's `urls.py`:

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

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

---

## Part 6: Using Templates and Django Template Inheritance

### Step 8: Create Blog Templates

1. Create a folder `templates/blog/` in the `blog` app directory.
2. Create the base template `base.html`:

```
html
Copy code
<!DOCTYPE html>
<html>
<head>
    <title>My Blog</title>
</head>
<body>
    <header>
        <h1>My Blog</h1>
        <nav>
            <a href="{% url 'post_list' %}">Home</a>
        </nav>
    </header>
    <div>
        {% block content %}
        {% endblock %}
    </div>
</body>
</html>
```

3. Create the template for listing blog posts, `post_list.html`, which extends `base.html`:

```
html
Copy code
{% extends 'blog/base.html' %}

{% block content %}
    <h2>Blog Posts</h2>
    <ul>
        {% for post in posts %}
        <li>
            <a href="{% url 'post_detail' post.id %}">{{ post.title }}</a>
```

```
            <p>Published on {{ post.pub_date }}</p>
        </li>
        {% endfor %}
    </ul>
{% endblock %}
```

4. Create the template for a single post, `post_detail.html`:

```
html
Copy code
{% extends 'blog/base.html' %}

{% block content %}
    <h2>{{ post.title }}</h2>
    <p>By {{ post.author.name }}</p>
    <p>{{ post.content }}</p>
    <p>Published on {{ post.pub_date }}</p>
{% endblock %}
```

---

## Part 7: Reversing URLs

### Step 9: Use URL Reversing in Templates

In the `post_list.html` template, you already used `{% url 'post_detail' post.id %}` to reverse the URL for each blog post. This ensures that you can change your URL patterns later without breaking the links in your templates.

---

## Bonus: Adding HTML Escaping and Filters

1. **HTML Escaping**: Django automatically escapes HTML in templates to prevent XSS attacks. Test this by adding HTML in the content of a post, and notice how it is safely escaped.
2. **Filters**: Use Django template filters in `post_detail.html` to format the `pub_date`:

   ```
   html
   Copy code
   <p>Published on {{ post.pub_date|date:"F d, Y" }}</p>
   ```

---

## Summary of Learning Objectives:

By the end of this exercise, students will have learned:

- How to set up a Django project and app.
- How to define **Django models** and handle migrations.
- How to use the **Django admin** interface to manage content.

- How to create **Django views** and **templates**.
- How to implement **URL routing** and use the **Django template language**.
- How to use **template inheritance** and **reversing URLs**.
- How to protect against XSS with **HTML escaping**.