Perfect! Let's go step by step on **how to apply URLs and nested URLs in Django**. I'll cover **basic URLs, app-level URLs, and nested URLs with examples**, including DRF if you want to make API endpoints.

1️⃣ Basic URLs in Django

1. **Project `urls.py`**
   All requests first go through this file.

```
# project/urls.py
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('students/', include('students.urls')),  # include app URLs
]
```

2. **App `urls.py`**
   Create a `urls.py` inside your app (`students`).

```
# students/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('', views.student_list, name='student-list'),       # /students/
    path('<int:id>/', views.student_detail, name='student-detail'),  # /students/1/
]
```

3. **Views**

```
# students/views.py
from django.http import HttpResponse

def student_list(request):
    return HttpResponse("List of students")

def student_detail(request, id):
    return HttpResponse(f"Student {id}")
```

✅ Now `/students/` → list, `/students/1/` → detail page.

2️⃣ Nested URLs (Child resources)

Suppose **Books belong to an Author**: `/authors/<id>/books/`

1. **Project `urls.py`**

```
path('authors/', include('authors.urls')),
```

2. **Author App `urls.py`**

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.author_list, name='author-list'),            # /authors/
    path('<int:author_id>/', views.author_detail, name='author-detail'), # /authors/1/
    path('<int:author_id>/books/', views.author_books, name='author-books'),  # nested URL
]
```

3. **Views**

```
def author_books(request, author_id):
    return HttpResponse(f"Books of author {author_id}")
```

✅ Now `/authors/1/books/` → list of books for author 1.

3️⃣ Nested URLs with DRF and ViewSets

Using **DRF routers**, nested resources are easier to manage.

1. Install **drf-nested-routers**

```
pip install drf-nested-routers
```

2. **ViewSets**

```python
from rest_framework import viewsets
from .models import Author, Book
from .serializers import AuthorSerializer, BookSerializer

class AuthorViewSet(viewsets.ModelViewSet):
    queryset = Author.objects.all()
    serializer_class = AuthorSerializer

class BookViewSet(viewsets.ModelViewSet):
    serializer_class = BookSerializer

    def get_queryset(self):
        return Book.objects.filter(author_id=self.kwargs['author_pk'])
```

3. **Routers**

```python
from rest_framework_nested import routers
from .views import AuthorViewSet, BookViewSet

router = routers.SimpleRouter()
router.register(r'authors', AuthorViewSet)

nested_router = routers.NestedSimpleRouter(router, r'authors',
lookup='author')
nested_router.register(r'books', BookViewSet, basename='author-books')

urlpatterns = router.urls + nested_router.urls
```

✅ Now DRF endpoints:

- `/authors/` → all authors
- `/authors/1/` → author detail
- `/authors/1/books/` → books for author 1
- `/authors/1/books/5/` → specific book of author 1

4️⃣ Summary

URL Type	Django Path Example	Notes
Basic	`/students/`	List or create students
Detail	`/students/<id>/`	Student detail
Nested/Child	`/authors/<id>/books/`	Books of a specific author
DRF Nested	`/authors/<author_pk>/books/`	Works with ViewSets & nested routers

✅ Rules of Thumb

1. Use **app-level** `urls.py` for modularity.
2. For **nested resources**, use `<parent_id>/<child>/` pattern.
3. In DRF, use **drf-nested-routers** for clean nested ViewSets.
4. Avoid deep nesting (>2 levels) to keep URLs readable.

Perfect! Let's create a **comprehensive nested router example** using Django REST Framework and **drf-nested-routers** that shows **multiple levels of nesting**, full CRUD, and realistic models.

We'll take a scenario like:

- **Author → Book → Chapter → Page**
- This is a **4-level nested structure** (Author is top-level, Page is the deepest).

1️⃣ Models

```
from django.db import models

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

class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.ForeignKey(Author, related_name='books',
on_delete=models.CASCADE)

class Chapter(models.Model):
    title = models.CharField(max_length=200)
    book = models.ForeignKey(Book, related_name='chapters',
on_delete=models.CASCADE)

class Page(models.Model):
    number = models.IntegerField()
    content = models.TextField()
    chapter = models.ForeignKey(Chapter, related_name='pages',
on_delete=models.CASCADE)
```

2️⃣ Serializers

```
from rest_framework import serializers
from .models import Author, Book, Chapter, Page

class PageSerializer(serializers.ModelSerializer):
    class Meta:
        model = Page
        fields = ['id', 'number', 'content', 'chapter']

class ChapterSerializer(serializers.ModelSerializer):
    pages = PageSerializer(many=True, read_only=True)

    class Meta:
        model = Chapter
        fields = ['id', 'title', 'book', 'pages']

class BookSerializer(serializers.ModelSerializer):
    chapters = ChapterSerializer(many=True, read_only=True)

    class Meta:
        model = Book
        fields = ['id', 'title', 'author', 'chapters']

class AuthorSerializer(serializers.ModelSerializer):
    books = BookSerializer(many=True, read_only=True)

    class Meta:
        model = Author
        fields = ['id', 'name', 'books']
```

3️⃣ ViewSets

```
from rest_framework import viewsets
from .models import Author, Book, Chapter, Page
from .serializers import AuthorSerializer, BookSerializer,
ChapterSerializer, PageSerializer

class AuthorViewSet(viewsets.ModelViewSet):
    queryset = Author.objects.all()
    serializer_class = AuthorSerializer

class BookViewSet(viewsets.ModelViewSet):
    serializer_class = BookSerializer
```

```
    def get_queryset(self):
        return Book.objects.filter(author_id=self.kwargs['author_pk'])

class ChapterViewSet(viewsets.ModelViewSet):
    serializer_class = ChapterSerializer

    def get_queryset(self):
        return Chapter.objects.filter(book_id=self.kwargs['book_pk'])

class PageViewSet(viewsets.ModelViewSet):
    serializer_class = PageSerializer

    def get_queryset(self):
        return Page.objects.filter(chapter_id=self.kwargs['chapter_pk'])
```

4️⃣ Nested Routers

```
from rest_framework_nested import routers
from .views import AuthorViewSet, BookViewSet, ChapterViewSet, PageViewSet

# Top-level router
router = routers.SimpleRouter()
router.register(r'authors', AuthorViewSet)

# 1st level nested: books under authors
books_router = routers.NestedSimpleRouter(router, r'authors',
lookup='author')
books_router.register(r'books', BookViewSet, basename='author-books')

# 2nd level nested: chapters under books
chapters_router = routers.NestedSimpleRouter(books_router, r'books',
lookup='book')
chapters_router.register(r'chapters', ChapterViewSet, basename='book-
chapters')

# 3rd level nested: pages under chapters
pages_router = routers.NestedSimpleRouter(chapters_router, r'chapters',
lookup='chapter')
pages_router.register(r'pages', PageViewSet, basename='chapter-pages')

# Combine all URLs
urlpatterns = router.urls + books_router.urls + chapters_router.urls +
pages_router.urls
```

5️⃣ Example URLs Generated

URL	Action
`/authors/`	List/create authors
`/authors/1/`	Retrieve/update/delete author
`/authors/1/books/`	List/create books for author 1
`/authors/1/books/2/`	Retrieve/update/delete book 2
`/authors/1/books/2/chapters/`	List/create chapters for book 2
`/authors/1/books/2/chapters/3/`	Retrieve/update/delete chapter 3
`/authors/1/books/2/chapters/3/pages/`	List/create pages for chapter 3
`/authors/1/books/2/chapters/3/pages/4/`	Retrieve/update/delete page 4

6️⃣ Key Points

1. Each nested router passes **parent ID as `kwargs`** (`author_pk`, `book_pk`, `chapter_pk`).
2. In ViewSets, filter queryset using these **parent IDs** to maintain hierarchy.
3. You can go **as deep as needed**, but 3–4 levels is usually enough.
4. Nested routers automatically create **CRUD endpoints** for each level.
5. Use **MiniSerializers** or `read_only` for nested relationships to avoid recursion.

This is basically the **biggest nested router example** you can realistically use in a project.

If you want, I can make a **visual diagram showing Author → Book → Chapter → Page nested routers** with arrows and endpoints. It makes it very easy to understand the hierarchy.

Perfect! Let's create a **nested router example** for a **University Management System** using Django REST Framework (DRF) and **drf-nested-routers**. This will be realistic and similar in complexity to the previous example but tailored to a university scenario.

1️⃣ Models

We'll model the hierarchy as:

- **University → Faculty → Department → Course → Student**

```
from django.db import models

class University(models.Model):
    name = models.CharField(max_length=100)
    location = models.CharField(max_length=200)

class Faculty(models.Model):
    name = models.CharField(max_length=100)
    university = models.ForeignKey(University, related_name='faculties',
on_delete=models.CASCADE)

class Department(models.Model):
    name = models.CharField(max_length=100)
    faculty = models.ForeignKey(Faculty, related_name='departments',
on_delete=models.CASCADE)

class Course(models.Model):
    name = models.CharField(max_length=100)
    department = models.ForeignKey(Department, related_name='courses',
on_delete=models.CASCADE)

class Student(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField()
    courses = models.ManyToManyField(Course, related_name='students')
```

2️⃣ Serializers

```
from rest_framework import serializers
from .models import University, Faculty, Department, Course, Student

class StudentSerializer(serializers.ModelSerializer):
    class Meta:
        model = Student
        fields = ['id', 'name', 'email']

class CourseSerializer(serializers.ModelSerializer):
    students = StudentSerializer(many=True, read_only=True)

    class Meta:
```

```
        model = Course
        fields = ['id', 'name', 'department', 'students']

class DepartmentSerializer(serializers.ModelSerializer):
    courses = CourseSerializer(many=True, read_only=True)

    class Meta:
        model = Department
        fields = ['id', 'name', 'faculty', 'courses']

class FacultySerializer(serializers.ModelSerializer):
    departments = DepartmentSerializer(many=True, read_only=True)

    class Meta:
        model = Faculty
        fields = ['id', 'name', 'university', 'departments']

class UniversitySerializer(serializers.ModelSerializer):
    faculties = FacultySerializer(many=True, read_only=True)

    class Meta:
        model = University
        fields = ['id', 'name', 'location', 'faculties']
```

3️⃣ ViewSets

```
from rest_framework import viewsets
from .models import University, Faculty, Department, Course, Student
from .serializers import UniversitySerializer, FacultySerializer,
DepartmentSerializer, CourseSerializer, StudentSerializer

class UniversityViewSet(viewsets.ModelViewSet):
    queryset = University.objects.all()
    serializer_class = UniversitySerializer

class FacultyViewSet(viewsets.ModelViewSet):
    serializer_class = FacultySerializer

    def get_queryset(self):
        return
Faculty.objects.filter(university_id=self.kwargs['university_pk'])

class DepartmentViewSet(viewsets.ModelViewSet):
    serializer_class = DepartmentSerializer

    def get_queryset(self):
```

```
        return
Department.objects.filter(faculty_id=self.kwargs['faculty_pk'])

class CourseViewSet(viewsets.ModelViewSet):
    serializer_class = CourseSerializer

    def get_queryset(self):
        return
Course.objects.filter(department_id=self.kwargs['department_pk'])

class StudentViewSet(viewsets.ModelViewSet):
    serializer_class = StudentSerializer

    def get_queryset(self):
        course_id = self.kwargs.get('course_pk')
        if course_id:
            return Student.objects.filter(courses__id=course_id)
        return Student.objects.all()
```

4️⃣ Nested Routers

```
from rest_framework_nested import routers
from .views import UniversityViewSet, FacultyViewSet, DepartmentViewSet,
CourseViewSet, StudentViewSet

# Top-level router
router = routers.SimpleRouter()
router.register(r'universities', UniversityViewSet)

# Level 1: Faculties under University
faculty_router = routers.NestedSimpleRouter(router, r'universities',
lookup='university')
faculty_router.register(r'faculties', FacultyViewSet,
basename='university-faculties')

# Level 2: Departments under Faculty
department_router = routers.NestedSimpleRouter(faculty_router,
r'faculties', lookup='faculty')
department_router.register(r'departments', DepartmentViewSet,
basename='faculty-departments')

# Level 3: Courses under Department
course_router = routers.NestedSimpleRouter(department_router,
r'departments', lookup='department')
course_router.register(r'courses', CourseViewSet, basename='department-
courses')
```

```
# Level 4: Students under Course
student_router = routers.NestedSimpleRouter(course_router, r'courses',
lookup='course')
student_router.register(r'students', StudentViewSet, basename='course-
students')

# Combine all URLs
urlpatterns = router.urls + faculty_router.urls + department_router.urls +
course_router.urls + student_router.urls
```

5️⃣ Example URLs

URL	Action
`/universities/`	List/Create U
`/universities/1/`	Retrieve/Upd University 1
`/universities/1/faculties/`	List/Create F University 1
`/universities/1/faculties/2/`	Retrieve/Upd Faculty 2
`/universities/1/faculties/2/departments/`	List/Create Departments Faculty 2
`/universities/1/faculties/2/departments/3/`	Retrieve/Upd Department 3
`/universities/1/faculties/2/departments/3/courses/`	List/Create C under Depart
`/universities/1/faculties/2/departments/3/courses/4/`	Retrieve/Upd Course 4
`/universities/1/faculties/2/departments/3/courses/4/students/`	List/Create S Course 4
`/universities/1/faculties/2/departments/3/courses/4/students/5/`	Retrieve/Upd Student 5

6️⃣ Notes / Best Practices

1. Use `related_name` for reverse relationships to avoid confusion.

2. For deeply nested serializers, consider **MiniSerializers** to avoid recursion.
3. Filter queryset in each ViewSet using `kwargs` from nested router.
4. Nested routers give **clean RESTful endpoints** for hierarchical data.
5. Avoid nesting too deep (>4 levels) to keep API simple.

If you want, I can **draw a full visual hierarchy diagram** for this University Management System showing **universities → faculties → departments → courses → students** with all nested URLs and endpoints.

Do you want me to do that?