**Project Overview**

This Django app is about managing **Students**, their **Profiles**, the **Courses** they take, and the **Departments** that offer those courses.

**🗂️ Project Structure**

Assuming the app is called core, here's a simplified structure:

School\_project/

├── core/

│ ├── models.py

│ ├── views.py

│ ├── urls.py

│ ├── forms.py

│ ├── templates/

│ │ └── core/

│ │ ├── student\_list.html

│ │ ├── student\_form.html

│ │ └── student\_confirm\_delete.html

└── login.html

├── /

│ ├── settings.py

│ ├── urls.py

## Step-by-step Setup

### ****Create the Django project and app****

django-admin startproject school\_project

or

python –m django startproject school\_project

cd school\_project

python manage.py startapp core

### ****Add app to settings****

In school\_project/settings.py:

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'core',

]

**1. Introduction to Django Models and ORM**

Django models are Python classes that define your database schema. Django uses an **ORM** (Object Relational Mapper) to allow querying and manipulating database records using Python code, instead of raw SQL.

**Example:**

Student.objects.all() # Gets all students (a SELECT query behind the scenes)

In your app, these models exist:

* Student
* Profile
* Course
* Department

**2. Creating Models and Defining Fields**

**models.py Code:**

from django.db import models

class Department(models.Model):

    name = models.CharField(max\_length=100)

    def \_\_str\_\_(self):

        return self.name

class Course(models.Model):

    name = models.CharField(max\_length=100)

    credits = models.IntegerField()

    department = models.ForeignKey(Department, on\_delete=models.CASCADE)

    def \_\_str\_\_(self):

        return self.name

class Profile(models.Model):

    address = models.CharField(max\_length=255)

    phone = models.CharField(max\_length=15)

    def \_\_str\_\_(self):

        return f"{self.address} - {self.phone}"

class Student(models.Model):

    name = models.CharField(max\_length=100)

    age = models.IntegerField()

    profile = models.OneToOneField(Profile, on\_delete=models.CASCADE)

    courses = models.ManyToManyField(Course)

    def \_\_str\_\_(self):

        return self.name

**models.py Explanation:**

class Department(models.Model):

name = models.CharField(max\_length=100)

* A simple table for departments (e.g., "Computer Science").

**This is a table like:**

| **id** | **name** |
| --- | --- |
| 1 | Computer Science |

class Course(models.Model):

name = models.CharField(max\_length=100)

credits = models.IntegerField()

department = models.ForeignKey(Department, on\_delete=models.CASCADE)

* Each course has a **name**, **credits**, and belongs to **one department** (via ForeignKey).
* name: Course name.
* credits: Number of credit hours.
* department: ForeignKey creates a **Many-to-One** relation → many courses can belong to one department.
* on\_delete=models.CASCADE: If a department is deleted, its courses are deleted too.

class Profile(models.Model):

address = models.CharField(max\_length=255)

phone = models.CharField(max\_length=15)

* Each student has a **profile** with additional info.

class Student(models.Model):

name = models.CharField(max\_length=100)

age = models.IntegerField()

profile = models.OneToOneField(Profile, on\_delete=models.CASCADE)

courses = models.ManyToManyField(Course)

* A student has:
  + OneToOneField to Profile (1 student ↔ 1 profile)
  + ManyToManyField with Course (1 student ↔ many courses)

**3. Database Migrations: makemigrations and migrate**

After defining models, run:

python manage.py makemigrations

python manage.py migrate

* makemigrations generates migration files based on model changes.
* migrate applies those changes to the actual database (creates tables).

**4. Querying and Manipulating Data with the ORM (CRUD operations)**

**In views.py, you’re doing full CRUD:**

from django.shortcuts import render, redirect, get\_object\_or\_404

from .models import Student

from .forms import StudentForm

from django.contrib.auth.decorators import login\_required

from django.contrib.auth import logout

def logout\_view(request):

    logout(request)

    return redirect('login')  # Or any page you want after logout

@login\_required

def student\_list(request):

    students = Student.objects.all()

    return render(request, 'core/student\_list.html', {'students': students})

@login\_required

def student\_create(request):

    if request.method == 'POST':

        form = StudentForm(request.POST)

        if form.is\_valid():

            form.save()

            return redirect('student\_list')

    else:

        form = StudentForm()

    return render(request, 'core/student\_form.html', {'form': form})

@login\_required

def student\_update(request, pk):

    student = get\_object\_or\_404(Student, pk=pk)

    form = StudentForm(request.POST or None, instance=student)

    if form.is\_valid():

        form.save()

        return redirect('student\_list')

    return render(request, 'core/student\_form.html', {'form': form})

@login\_required

def student\_delete(request, pk):

    student = get\_object\_or\_404(Student, pk=pk)

    if request.method == 'POST':

        student.delete()

        return redirect('student\_list')

    return render(request, 'core/student\_confirm\_delete.html', {'student': student})

**Create a student:**

def student\_create(request):

...

form.save() # Saves to DB

* If form is submitted (POST), it validates and saves the new student.
* If not, it displays an empty form.

**Read students:**

def student\_list(request):

students = Student.objects.all() # SELECT \* FROM student

**Update student:**

def student\_update(request, pk):

student = get\_object\_or\_404(Student, pk=pk)

...

form.save()

* Finds a student by primary key (pk) and shows a pre-filled form.
* On submission, updates student info.

**Delete student:**

def student\_delete(request, pk):

student = get\_object\_or\_404(Student, pk=pk)

student.delete()

* Confirms deletion of the selected student.
* Deletes and redirects back to the list.

So, you’re using the Django ORM for all operations — no raw SQL!

**5. Relationships: ForeignKey, ManyToManyField, OneToOneField**

Your models demonstrate **all 3 major relationships** in Django:

| **Relationship Type** | **Example** | **Explanation** |
| --- | --- | --- |
| ForeignKey | Course → Department | Many Courses belong to one Department |
| ManyToManyField | Student ↔ Course | A student can enroll in many courses and vice versa |
| OneToOneField | Student → Profile | One student has one profile |

**forms.py**

from django import forms

from .models import Student, Profile, Course

class ProfileForm(forms.ModelForm):

    class Meta:

        model = Profile

        fields = ['address', 'phone']

class StudentForm(forms.ModelForm):

    class Meta:

        model = Student

        fields = ['name', 'age', 'profile', 'courses']

class CourseForm(forms.ModelForm):

    class Meta:

        model = Course

        fields = '\_\_all\_\_'

**URLs Configuration**

**core/urls.py:**

from django.urls import path

from . import views

urlpatterns = [

    path('', views.student\_list, name='student\_list'),

    path('student/create/', views.student\_create, name='student\_create'),

    path('student/update/<int:pk>/', views.student\_update, name='student\_update'),

    path('student/delete/<int:pk>/', views.student\_delete, name='student\_delete'),

]

* Maps URLs to the corresponding views:
  + / → list all students
  + /student/create/ → create a student
  + /student/update/1/ → update student with ID 1
  + /student/delete/1/ → confirm deletion

These URLs connect user requests to the corresponding views — enabling full CRUD functionality.

**School\_project/urls.py:**

from django.contrib import admin

from django.urls import path, include

from django.contrib.auth import views as auth\_views

from core.views import logout\_view

urlpatterns = [

    path('admin/', admin.site.urls),

    path('', include('core.urls')),

    path('login/', auth\_views.LoginView.as\_view(template\_name='core/login.html'), name='login'),

    path('logout/', logout\_view, name='logout'),

]

**Core/admin.py**

from django.contrib import admin

from .models import Student, Profile, Course, Department

admin.site.register(Student)

admin.site.register(Profile)

admin.site.register(Course)

admin.site.register(Department)

**Templates**

**core/templates/core/login.html**

<!-- templates/core/login.html -->

<h2>Login</h2>

<form method="post">

    {% csrf\_token %}

    {{ form.as\_p }}

    <button type="submit">Login</button>

</form>

**core/templates/core/student\_confirm\_delete.html**

<!-- core/templates/core/student\_confirm\_delete.html -->

<!DOCTYPE html>

<html>

<head>

    <title>Delete Student</title>

</head>

<body>

    <h1>Are you sure you want to delete "{{ student.name }}"?</h1>

    <form method="post">

        {% csrf\_token %}

        <button type="submit">✅ Confirm Delete</button>

        <a href="{% url 'student\_list' %}">↩️ Cancel</a>

    </form>

</body>

</html>

**core/templates/core/student\_form.html**

<!DOCTYPE html>

<html>

<head>

    <title>{% if form.instance.pk %}Edit{% else %}Add{% endif %} Student</title>

</head>

<body>

    <h1>{% if form.instance.pk %}Edit{% else %}Add{% endif %} Student</h1>

    <form method="post">

        {% csrf\_token %}

        {{ form.as\_p }}

        <button type="submit">💾 Save</button>

        <a href="{% url 'student\_list' %}">↩️ Cancel</a>

    </form>

</body>

</html>

**core/templates/core/student\_list.html**

<!DOCTYPE html>

<html>

<head>

    <title>Student List</title>

</head>

<body>

{% if request.user.is\_authenticated %}

    <p>Hello, {{ request.user.username }} | <a href="{% url 'logout' %}">Logout</a></p>

{% else %}

    <p><a href="{% url 'login' %}">Login</a></p>

{% endif %}

    <h1>Students</h1>

    <a href="{% url 'student\_create' %}">➕ Add Student</a>

    <hr>

    {% if students %}

        <ul>

            {% for student in students %}

                <li>

                    <strong>{{ student.name }}</strong> (Age: {{ student.age }})

                    <br>Profile: {{ student.profile.address }} | {{ student.profile.phone }}

                    <br>Courses:

                    {% for course in student.courses.all %}

                        {{ course.name }}{% if not forloop.last %}, {% endif %}

                    {% empty %}

                        No courses enrolled.

                    {% endfor %}

                    <br>

                    <a href="{% url 'student\_update' student.pk %}">✏️ Edit</a>

                    <a href="{% url 'student\_delete' student.pk %}">️ Delete</a>

                </li>

                <hr>

            {% endfor %}

        </ul>

    {% else %}

        <p>No students available.</p>

    {% endif %}

</body>

</html>

python manage.py createsuperuser

python manage.py migrate(optional)

**Settings Update**

In settings.py, set the login redirect:

LOGIN\_URL = 'login'

LOGIN\_REDIRECT\_URL = '/'

This makes login redirect users to / after successful login.

python manage.py runserver

**Open** [**url:-**](url:-)

[**http://127.0.0.1:8000/admin/**](http://127.0.0.1:8000/admin/)

**Commands:-**

**C:\Users\piyush>django-admin startproject school\_project**

**C:\Users\piyush>cd school\_project**

**C:\Users\piyush\school\_project>python manage.py startapp core**

**C:\Users\piyush\school\_project>python manage.py makemigrations**

**Migrations for 'core':**

**core\migrations\0001\_initial.py**

**+ Create model Department**

**+ Create model Profile**

**+ Create model Course**

**+ Create model Student**

**C:\Users\piyush\school\_project>python manage.py migrate**

**Operations to perform:**

**Apply all migrations: admin, auth, contenttypes, core, sessions**

**Running migrations:**

**Applying contenttypes.0001\_initial... OK**

**Applying auth.0001\_initial... OK**

**Applying admin.0001\_initial... OK**

**Applying admin.0002\_logentry\_remove\_auto\_add... OK**

**Applying admin.0003\_logentry\_add\_action\_flag\_choices... OK**

**Applying contenttypes.0002\_remove\_content\_type\_name... OK**

**Applying auth.0002\_alter\_permission\_name\_max\_length... OK**

**Applying auth.0003\_alter\_user\_email\_max\_length... OK**

**Applying auth.0004\_alter\_user\_username\_opts... OK**

**Applying auth.0005\_alter\_user\_last\_login\_null... OK**

**Applying auth.0006\_require\_contenttypes\_0002... OK**

**Applying auth.0007\_alter\_validators\_add\_error\_messages... OK**

**Applying auth.0008\_alter\_user\_username\_max\_length... OK**

**Applying auth.0009\_alter\_user\_last\_name\_max\_length... OK**

**Applying auth.0010\_alter\_group\_name\_max\_length... OK**

**Applying auth.0011\_update\_proxy\_permissions... OK**

**Applying auth.0012\_alter\_user\_first\_name\_max\_length... OK**

**Applying core.0001\_initial... OK**

**Applying sessions.0001\_initial... OK**

**C:\Users\piyush\school\_project>python manage.py createsuperuser**

**Username (leave blank to use 'piyush'):**

**Email address:**

**Password:**

**Password (again):**

**Superuser created successfully.**

**C:\Users\piyush\school\_project>python manage.py migrate**

**Operations to perform:**

**Apply all migrations: admin, auth, contenttypes, core, sessions**

**Running migrations:**

**No migrations to apply.**

**C:\Users\piyush\school\_project>python manage.py runserver**

**Watching for file changes with StatReloader**

**Performing system checks...**

**System check identified no issues (0 silenced).**

**September 01, 2025 - 22:06:19**

**Django version 5.2.5, using settings 'school\_project.settings'**

**Starting development server at http://127.0.0.1:8000/**

**Quit the server with CTRL-BREAK.**