Django Student Management Mini Project

🎯 Project Agenda & Steps

1️⃣ **Create Project & App** – `django-admin startproject`, `startapp`
2️⃣ **Register App** – Add to `INSTALLED_APPS`
3️⃣ **Design Model** – Define `Student` model
4️⃣ **Migrate DB** – `makemigrations` + `migrate`
5️⃣ **Admin Panel** – Create superuser, register model, customize admin
6️⃣ **Create Views** – Show data in HTML & JSON
7️⃣ **Configure URLs** – Connect views to routes
8️⃣ **Design Template** – `std.html` to display student info
9️⃣ **Connect MySQL** – Setup MySQL DB in `settings.py`
🔟 **Run Server** – `python manage.py runserver`

🧩 Key Features

📦 ORM Queries – Fetch/Filter student data
📋 Admin Panel – Manage student records easily
🌐 HTML Page – Display data dynamically
🛠️ JSON API – Provide data in JSON format

💡 Bonus

📘 **MySQL Setup** – Use MySQL as your DB
🧠 **Cheat Sheet** – Common ORM & MySQL commands
❓ **FAQs** – Learn key Django terms like ORM, migrate, JsonResponse
💂 **Admin Customization** – `list_display`, `search_fields`, `list_filter`

Django Student Management Mini Project

Steps:

Step	Command / Action	Description
1	django-admin startproject studentproject	Create main project
2	cd studentproject	Navigate into project folder
3	python manage.py startapp testapp	Create app named testapp
4	Add 'testapp' in INSTALLED_APPS	Enable app in settings.py
5	Define Student model in models.py	Define DB structure
6	python manage.py makemigrations	Generate migration file
7	python manage.py migrate	Create tables in DB
8	python manage.py createsuperuser	Login to admin panel
9	Register model in admin.py	Manage student data from admin
10	Create view in views.py	Fetch and send data to template
11	Add app & project-level URLs	Route to views
12	Create template std.html	Show student data
13	python manage.py runserver	Start development server

💻 1. Install MySQL Server & Command Line Tool

OS	Installation Guide
Windows	☞ Download from official site:https://dev.mysql.com/downloads/installer/ ✓ Choose **MySQL Installer for Windows** ✓ Select **Developer Default** ✓ Install **MySQL Server + MySQL Shell + MySQL Command Line Tool**
Mac	Use Homebrew:`brew install mysql`

After install:

✓ Open **MySQL Command Line Tool**

✓ Login using:

⚙ Django Settings to Connect with MySQL

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'studentdb',
        'USER': 'root',
        'PASSWORD': 'root',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

Django ORM Command	Equivalent MySQL Query
`Student.objects.all()`	`SELECT * FROM student;`
`Student.objects.filter(name='Ali')`	`SELECT * FROM student WHERE name='Ali';`
Student.objects.order_by('-marks')	SELECT * FROM student ORDER BY marks DESC;
Student.objects.values()	SELECT name, rollno... FROM student;
Student.objects.get(pk=1)	SELECT * FROM student WHERE id=1;

Commonly Used MySQL Commands (Cheat Sheet)

Category	Command	Purpose
Login	mysql -u root -p	Log in to MySQL as root user
Show Databases	SHOW DATABASES;	List all available databases
Create Database	CREATE DATABASE dbname;	Create a new database
Use Database	USE dbname;	Switch to a database
Drop Database	DROP DATABASE dbname;	Delete a database
Show Tables	SHOW TABLES;	List all tables in the selected DB
Create Table	CREATE TABLE table_name (id INT, name VARCHAR(50));	Create a new table
Describe Table	DESCRIBE table_name;	Show table structure
Drop Table	DROP TABLE table_name;	Delete a table
Insert Data	INSERT INTO table_name VALUES (1, 'John');	Insert a row
Select All	SELECT * FROM table_name;	View all rows from a table
Select Columns	SELECT name FROM table_name;	View specific column(s)
Where Clause	SELECT * FROM table_name WHERE id = 1;	Filter records
Order By	SELECT * FROM table_name ORDER BY name ASC;	Sort results

🔢 Category	🛠 Command	📄 Purpose
Update Record	UPDATE table_name SET name = 'Ali' WHERE id = 1;	Update data
Delete Record	DELETE FROM table_name WHERE id = 1;	Delete specific row
Count Records	SELECT COUNT(*) FROM table_name;	Count total records
Show Current DB	SELECT DATABASE();	Show currently selected database
Show Date/Time	SELECT NOW();	Get current date and time
Exit MySQL	EXIT;	Logout from MySQL

📁 **testapp/models.py**

```
from django.db import models

class Student(models.Model):
    rollno = models.IntegerField()
    name = models.CharField(max_length=30)
    dob = models.DateField()
    marks = models.IntegerField()
    email = models.EmailField()
    phonenumber = models.BigIntegerField()
    address = models.TextField()


    def __str__(self):
        return f"{self.rollno} - {self.name}"
```

Note:

The __str__ method defines **how the object looks when printed** (like in admin panel or shell).

```
return f"{self.rollno} - {self.name}"
```

means it will show **"101 - John"** instead of **"Student object (1)"**.

admin.py

```
from django.contrib import admin

from .models import Student

class StudentAdmin(admin.ModelAdmin):

    # Show these columns in admin list view

    list_display = ['rollno', 'name', 'marks', 'email']


    # Add search box (by name or roll number)

    search_fields = ['name', 'rollno']


    # Add filter sidebar (by marks)

    list_filter = ['marks']


    # Sort by roll number by default

    ordering = ['rollno']
```

```
# Register the model and admin class

admin.site.register(Student, StudentAdmin)
```

2. Views (HTML + JSON Response)

testapp/views.py

```
from django.shortcuts import render
from django.http import JsonResponse
from testapp.models import Student

def student_view(request):
    student_list = Student.objects.all().order_by('-marks')
    return render(request, 'testapp/std.html', {'student_list': student_list})

def student_json_view(request):
    data = list(Student.objects.values())
    return JsonResponse(data, safe=False)
```

3. URLs

studentproject/urls.py

```
from django.contrib import admin
from django.urls import path, include

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

testapp/urls.py

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

urlpatterns = [
   path('students/', views.student_view, name='student-view'),
   path('api/students/', views.student_json_view, name='student-json'),
]
```

4. HTML Template (std.html)

```
<!DOCTYPE html>
{% load static %}
<html>
<head>
 <title>Student Info</title>
 <link rel="stylesheet" href="{% static 'css/std.css' %}">
</head>
<body>
 <h1>Student Information</h1>
 {% if student_list %}
  {% for student in student_list %}
   <h2>{{ student.name }}</h2>
   <ul>
    <li>Roll No: {{ student.rollno }}</li>
    <li>DOB: {{ student.dob }}</li>
    <li>Marks: {{ student.marks }}</li>
    <li>Email: {{ student.email }}</li>
```

```
      <li>Phone: {{ student.phonenumber }}</li>
      <li>Address: {{ student.address }}</li>
    </ul>
    <hr>
  {% endfor %}
 {% else %}
  <p>No student data found.</p>
 {% endif %}
</body>
</html>
```

5. ORM Commands in Tabular Format

Command	Description	Example
.all()	Fetch all records	Student.objects.all()
.filter()	Apply condition	Student.objects.filter(marks__lt=35)
.exclude()	Exclude matching	Student.objects.exclude(name='Ali')
.order_by()	Sort records	Student.objects.order_by('-marks')
.values()	Return dict-like data	Student.objects.values()
.get()	Return single object	Student.objects.get(rollno=101)
.count()	Total entries	Student.objects.count()
.first() / .last()	First / Last entry	Student.objects.first()

6. FAQs

Question	Answer
What is ORM in Django?	ORM (Object Relational Mapper) lets you interact with DB using Python code instead of SQL.
What is makemigrations?	It creates migration files to prepare DB structure from models.
What is migrate?	It applies migration files to the actual database.

How do you show data in admin panel?	Register your model using admin.site.register().
How to return JSON data?	Use JsonResponse() with values() or values_list().

7. Tricky Concept

ORM works like a translator between Python code and SQL queries.

Example: Student.objects.filter(marks__lt=35) → SQL: SELECT * FROM student WHERE marks < 35;

📚 Django Admin Panel – Quick Notes for Students

📌 What is `admin.py`?

- A file used to **register models** and **customize** how data is shown in Django's admin panel.
- Django gives you a **built-in admin interface** to manage database records.

✅ Why Register a Model?

Without registering the model, it won't appear in the admin panel.

```
from django.contrib import admin
from .models import Student

admin.site.register(Student)
```

💡 What is an Admin Class?

A **custom class** used to **control how data is displayed** in the admin panel (like columns, search, filters, etc.)

🔍 Example:

```
from django.contrib import admin
from .models import Student
```

```
class StudentAdmin(admin.ModelAdmin):
    list_display = ['rollno', 'name', 'marks']  # Show these columns
    search_fields = ['name']                    # Search box
    list_filter = ['marks']                     # Filter options

admin.site.register(Student, StudentAdmin)
```

Field Use (Tabular Format)

Feature	Purpose
`list_display`	Shows selected columns in table view
`search_fields`	Adds a search bar for given fields
`list_filter`	Adds a sidebar filter
`ordering`	Sets default sort order
`readonly_fields`	Makes fields uneditable

⚠ Without Admin Class vs With Admin Class

Without Admin Class	With Admin Class
Basic view (just object)	Shows table with multiple columns
No search or filter	Adds search bar and filter options
Hard to manage large data	Easy navigation and clean data handling

---

? Key Points:

- The admin panel is like a **control center** for your database.
- `admin.ModelAdmin` helps you make it user-friendly.
- Doesn't affect what's saved in the database — only how it's shown.

🎯 In Short:

`admin.py` + Admin Class = Clean & Powerful Admin Panel

📌 What it does:

Feature	What It Helps With
`list_display`	Shows selected columns in table
`search_fields`	Allows searching by name/rollno
`list_filter`	Adds filter on right side (marks)
`ordering`	Sorts rows by roll number

✅ Output:

This will make the admin panel:

- Easy to view
- Easy to search
- Easy to filter
- Easy to manage