**🎯 Learning Goals**

1. Build a Django project end‑to‑end (models → views → templates).
2. Understand CRUD operations with ListView / CreateView / UpdateView / DeleteView.
3. See live traffic simulation (users‑trying, cache, server‑load) and learn basic caching.
4. Seed dummy data & test real‑world load.

**🗓️ 5‑Day Classroom Road‑map**

| **Day** | **Topics & Activities** | **Homework / Lab** |
| --- | --- | --- |
| **1** | • Python + virtual‑env setup • django-admin startproject result\_portal . • Create app portal & add to INSTALLED\_APPS | Install VS Code, run *Hello Django* page |
| **2** | • **Models** Student, Result – explain OneToOneField, unique=True • makemigrations vs migrate | Draw ER diagram; add one student via Django admin |
| **3** | • **Forms**: plain LoginForm vs ModelForm • **Views**: function‑based login/result & stats helper • URL routing basics | Modify login page style (Bootstrap colors) |
| **4** | • **Class‑based CRUD views** – L/C/U/D for Student & Result • **Templates** inheritance (base.html) & pagination • Navbar links | Add search bar to student list |
| **5** | • Live **Jam Dashboard** – JSON AJAX polling, progress‑bar update • Caching (LocMemCache / Redis), why & how • **generate\_dummy\_data** command; run hey or locust for load | Short quiz + deploy to Heroku / Render (optional) |

**🛠️ Detailed Walk‑through**

**1️⃣ Environment & Project**

bash

CopyEdit

python -m venv .venv

.\.venv\Scripts\activate

pip install django faker

django-admin startproject result\_portal .

python manage.py startapp portal

*Explain:* why we use virtual‑env, project vs app folders.

**2️⃣ Models (portal/models.py)**

python

CopyEdit

class Student(models.Model):

register\_number = models.CharField(max\_length=20, unique=True)

dob = models.DateField()

class Result(models.Model):

student = models.OneToOneField(Student, on\_delete=models.CASCADE)

semester = models.PositiveSmallIntegerField()

gpa = models.DecimalField(max\_digits=4, decimal\_places=2)

*Teach:*

* OneToOneField similar to “PK–FK (1‑1)” in DB.
* unique=True prevents duplicates → shows IntegrityError if violated.

Run:

bash

CopyEdit

python manage.py makemigrations

python manage.py migrate

**3️⃣ Forms**

python

CopyEdit

class LoginForm(forms.Form): # simple form, not tied to DB

class StudentForm(ModelForm): # CRUD – auto build fields

class ResultForm(ModelForm):

*Point out*: Difference between plain Form (manual fields) vs ModelForm (auto‑sync with model).

**4️⃣ Views**

**🔹 Traffic simulator helper**

python

CopyEdit

def \_simulate\_stats():

users = cache.get\_or\_set("users\_trying", 0, timeout=10)

cache.incr("users\_trying")

load = random.randint(15,90)

return {...}

Explain cache counters, why timeout=10 auto‑expires (avoids infinite growth).

**🔹 Login flow**

1. POST → validate LoginForm.
2. Try cache for student key.
3. If DB hit → store session sid.
4. Redirect to result page.

*Demo:* print cache\_hit in console; watch effect after second login.

**🔹 Class‑based CRUD**

Show code for StudentList, StudentCreate, etc.  
Explain generic view attributes (model, template\_name, success\_url).

**5️⃣ Templates (Bootstrap)**

*Key ideas:*

* {% extends "base.html" %} → DRY layout.
* Pagination block uses page\_obj.
* jam\_dashboard.html uses JS fetch every 3 s → live update.

Classroom demo: change progress‑bar color by adding:

html

CopyEdit

{% if server\_load >= 50 %}

bar.classList.remove("bg-success"); bar.classList.add("bg-danger");

{% endif %}

**6️⃣ URL Routing**

Walk through:

python

CopyEdit

path("", views.login\_view) # root

path("jam/", views.jam\_dashboard) # live stats

path("students/", StudentList.as\_view()) # CRUD group

Explain app\_name = "portal" and reverse lookup {% url 'portal:student\_add' %}.

**7️⃣ Seeding Data & Load Test**

bash

CopyEdit

python manage.py generate\_dummy\_data

hey -n 500 -c 50 http://127.0.0.1:8000/

Students observe Jam Dashboard spiking.

*Concepts:* concurrency (-c 50), total requests (-n 500).

**🎓 Evaluation**

* Mini‑quiz: “Why do we cache login lookups?”
* Hands‑on: Add search query param to StudentList (filter register\_number\_\_icontains).
* Stretch: deploy to Render; configure Postgres + Redis.

### 🔧 “Student Jamming Portal” ‑‑ integrating the live‑load simulation ****into the CRUD site****

Below you’ll add **one extra page** ( /jam/ ) that always shows the real‑time “Users Trying / Routed to / Server Load / Status” panel, plus a quick link in the navbar so staff can open it while they manage students & results.

## 1️⃣ views.py – add a dedicated ****jam\_dashboard**** view

python

CopyEdit

# (keep existing imports + code)

def jam\_dashboard(request):

"""Standalone dashboard that refreshes every 3 s via JS."""

stats = \_simulate\_stats()

return render(request, "portal/jam\_dashboard.html", stats)

## 2️⃣ urls.py – route for the new page

python

CopyEdit

urlpatterns = [

# existing login, result & CRUD paths …

path("jam/", views.jam\_dashboard, name="jam\_dashboard"),

]

## 3️⃣ Template: jam\_dashboard.html

html

CopyEdit

{% extends "base.html" %}

{% block title %}Jam Dashboard{% endblock %}

{% block content %}

<div class="container" id="stats-box" style="max-width:400px">

<div class="card shadow p-4 mt-4">

<h4 class="text-center mb-3">Live Traffic – Jamming Simulation</h4>

<p class="mb-1"><strong>Users Trying:</strong> <span id="users">{{ users\_trying }}</span></p>

<p class="mb-1"><strong>Routed to:</strong> <span id="route">{{ routed\_to }}</span></p>

<div class="progress mb-2" style="height:8px">

<div id="bar" class="progress-bar bg-success" style="width: {{ server\_load }}%"></div>

</div>

<p class="mb-0"><strong>Server Load:</strong> <span id="load">{{ server\_load }}</span>%</p>

<p class="mb-0"><strong>Status:</strong> <span id="status">{{ status }}</span></p>

</div>

</div>

<!-- Auto‑refresh using fetch every 3 s -->

<script>

setInterval(() => {

fetch("{% url 'portal:jam\_dashboard' %}?ajax=1")

.then(r => r.json()) // when ?ajax=1 we’ll return JSON (see below)

.then(d => {

document.getElementById("users").textContent = d.users\_trying;

document.getElementById("route").textContent = d.routed\_to;

document.getElementById("load").textContent = d.server\_load;

document.getElementById("status").textContent= d.status;

document.getElementById("bar").style.width = d.server\_load + "%";

});

}, 3000);

</script>

{% endblock %}

## 4️⃣ Make the view return ****JSON**** for ?ajax=1

Add this small change inside **jam\_dashboard**:

python

CopyEdit

from django.http import JsonResponse

def jam\_dashboard(request):

stats = \_simulate\_stats()

if request.GET.get("ajax"):

return JsonResponse(stats)

return render(request, "portal/jam\_dashboard.html", stats)

## 5️⃣ Navbar link – update base.html

html

CopyEdit

<nav class="navbar navbar-expand-sm navbar-light bg-white shadow-sm mb-4 px-3">

<a class="navbar-brand fw-bold" href="/">Result Portal</a>

<div class="navbar-nav">

<a class="nav-link" href="{% url 'portal:student\_list' %}">Students</a>

<a class="nav-link" href="{% url 'portal:result\_list' %}">Results</a>

<a class="nav-link text-danger fw-semibold" href="{% url 'portal:jam\_dashboard' %}">Jam Dashboard</a>

</div>

</nav>

### 💡 How it works

* **Normal page load** → renders current stats.
* **JavaScript** polls ?ajax=1 every 3 s → backend returns fresh JSON without full HTML render → DOM updates smoothly.
* Teachers / admins can keep the **Jam Dashboard** open in a tab while editing students & results.

### 🗣️ Tamil Quick Recap

1. **jam\_dashboard view** – \_simulate\_stats() call பண்ணி stats JSON‑ஆவும், HTML‑ஆவும் return பண்ணும்.
2. **jam\_dashboard.html** – card UI + JS auto‑refresh (3 s).
3. **urls.py** – path("jam/", …) add பண்ணு.
4. **base.html nav‑ல்** Jam Dashboard link add செய்தால் CRUD பக்கத்திலிருந்தே open பண்ணலாம்.

இவ்வளவு copy‑paste செய்த பிறகு python manage.py runserver → /jam/ visit பண்ணிப் பாருங்கள்; progress‑bar & numbers auto‑update ஆகும்! 👍

Ask ChatGPT

##### You said:

full code sollu

###### ChatGPT said:

Below is a **complete, copy‑pastable “Student Result + Jamming Portal”** project.  
Everything is grouped by file path so you can drop each block into the right place and run.

**Project layout**

markdown

CopyEdit

result\_portal/ ← django‑admin startproject result\_portal .

├── manage.py

├── result\_portal/ ← project settings folder

│ ├── \_\_init\_\_.py

│ ├── settings.py

│ ├── urls.py

│ └── …

└── portal/ ← python manage.py startapp portal

├── \_\_init\_\_.py

├── admin.py

├── apps.py

├── models.py

├── forms.py

├── views.py

├── urls.py

├── management/

│ └── commands/

│ └── generate\_dummy\_data.py

└── templates/

└── portal/

├── base.html

├── login.html

├── result.html

├── jam\_dashboard.html

├── student\_list.html

├── student\_form.html

├── student\_confirm\_delete.html

├── result\_list.html

├── result\_form.html

└── result\_confirm\_delete.html

## 1️⃣ portal/models.py

python

CopyEdit

from django.db import models

class Student(models.Model):

register\_number = models.CharField(max\_length=20, unique=True)

dob = models.DateField()

def \_\_str\_\_(self):

return self.register\_number

class Result(models.Model):

student = models.OneToOneField(Student, on\_delete=models.CASCADE)

semester = models.PositiveSmallIntegerField()

gpa = models.DecimalField(max\_digits=4, decimal\_places=2)

updated = models.DateTimeField(auto\_now=True)

def \_\_str\_\_(self):

return f"{self.student} – SEM {self.semester}"

## 2️⃣ portal/forms.py

python

CopyEdit

from django import forms

from .models import Student, Result

class LoginForm(forms.Form):

register\_number = forms.CharField(label="Register Number")

dob = forms.DateField(label="DOB")

class StudentForm(forms.ModelForm):

class Meta:

model = Student

fields = ["register\_number", "dob"]

class ResultForm(forms.ModelForm):

class Meta:

model = Result

fields = ["student", "semester", "gpa"]

## 3️⃣ portal/views.py

python

CopyEdit

import random, datetime

from django.shortcuts import render, redirect

from django.urls import reverse\_lazy

from django.core.cache import cache

from django.http import JsonResponse

from django.views.generic import (

ListView, CreateView, UpdateView, DeleteView

)

from .forms import LoginForm, StudentForm, ResultForm

from .models import Student, Result

# ---------- traffic‑sim helper ----------

def \_simulate\_stats():

users = cache.get\_or\_set("users\_trying", 0, timeout=10)

cache.incr("users\_trying")

load = random.randint(15, 90)

return {

"users\_trying": users,

"routed\_to" : f"Server {random.randint(1, 2)}",

"server\_load" : load,

"status" : "High Load" if load >= 50 else "Normal",

}

# ---------- login & result ----------

def login\_view(request):

stats, cache\_hit = \_simulate\_stats(), None

if request.method == "POST":

form = LoginForm(request.POST)

if form.is\_valid():

reg, dob = form.cleaned\_data["register\_number"], form.cleaned\_data["dob"]

key = f"student:{reg}"

student = cache.get(key)

cache\_hit = bool(student)

if not student:

student = Student.objects.filter(register\_number=reg, dob=dob).first()

if student:

cache.set(key, student, timeout=60)

if student:

request.session["sid"] = student.id

return redirect("portal:result")

form.add\_error(None, "Invalid credentials")

else:

form = LoginForm()

return render(request, "portal/login.html",

{"form": form, "cache\_hit": cache\_hit, \*\*stats})

def result\_view(request):

sid = request.session.get("sid")

if not sid:

return redirect("portal:login")

student = Student.objects.get(id=sid)

result = Result.objects.filter(student=student).first()

return render(request, "portal/result.html",

{"student": student, "result": result,

"now": datetime.datetime.now()})

# ---------- jam dashboard ----------

def jam\_dashboard(request):

stats = \_simulate\_stats()

if request.GET.get("ajax"):

return JsonResponse(stats)

return render(request, "portal/jam\_dashboard.html", stats)

# ---------- Student CRUD ----------

class StudentList(ListView):

model, template\_name, paginate\_by = Student, "portal/student\_list.html", 20

class StudentCreate(CreateView):

model, form\_class, template\_name = Student, StudentForm, "portal/student\_form.html"

success\_url = reverse\_lazy("portal:student\_list")

class StudentUpdate(UpdateView):

model, form\_class, template\_name = Student, StudentForm, "portal/student\_form.html"

success\_url = reverse\_lazy("portal:student\_list")

class StudentDelete(DeleteView):

model, template\_name = Student, "portal/student\_confirm\_delete.html"

success\_url = reverse\_lazy("portal:student\_list")

# ---------- Result CRUD ----------

class ResultList(ListView):

model, template\_name, paginate\_by = Result, "portal/result\_list.html", 20

class ResultCreate(CreateView):

model, form\_class, template\_name = Result, ResultForm, "portal/result\_form.html"

success\_url = reverse\_lazy("portal:result\_list")

class ResultUpdate(UpdateView):

model, form\_class, template\_name = Result, ResultForm, "portal/result\_form.html"

success\_url = reverse\_lazy("portal:result\_list")

class ResultDelete(DeleteView):

model, template\_name = Result, "portal/result\_confirm\_delete.html"

success\_url = reverse\_lazy("portal:result\_list")

## 4️⃣ portal/urls.py

python

CopyEdit

from django.urls import path

from . import views

app\_name = "portal"

urlpatterns = [

# login & result

path("", views.login\_view, name="login"),

path("home/", views.result\_view, name="result"),

# jam dashboard

path("jam/", views.jam\_dashboard, name="jam\_dashboard"),

# Student CRUD

path("students/", views.StudentList.as\_view(), name="student\_list"),

path("students/add/", views.StudentCreate.as\_view(), name="student\_add"),

path("students/<int:pk>/edit/", views.StudentUpdate.as\_view(), name="student\_edit"),

path("students/<int:pk>/delete/",views.StudentDelete.as\_view(), name="student\_delete"),

# Result CRUD

path("results/", views.ResultList.as\_view(), name="result\_list"),

path("results/add/", views.ResultCreate.as\_view(), name="result\_add"),

path("results/<int:pk>/edit/", views.ResultUpdate.as\_view(), name="result\_edit"),

path("results/<int:pk>/delete/", views.ResultDelete.as\_view(), name="result\_delete"),

]

Add to **project** result\_portal/urls.py:

python

CopyEdit

from django.urls import path, include

urlpatterns = [ path("", include("portal.urls")) ]

## 5️⃣ Templates (portal/templates/portal/\*.html)

**base.html**

html

CopyEdit

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8"><title>{% block title %}Result Portal{% endblock %}</title>

<link rel="stylesheet"

href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css">

</head>

<body class="bg-light">

<nav class="navbar navbar-expand-sm navbar-light bg-white shadow-sm mb-4 px-3">

<a class="navbar-brand fw-bold" href="/">Result Portal</a>

<div class="navbar-nav">

<a class="nav-link" href="{% url 'portal:student\_list' %}">Students</a>

<a class="nav-link" href="{% url 'portal:result\_list' %}">Results</a>

<a class="nav-link text-danger fw-semibold" href="{% url 'portal:jam\_dashboard' %}">

Jam Dashboard

</a>

</div>

</nav>

{% block content %}{% endblock %}

</body>

</html>

**login.html**

html

CopyEdit

<!DOCTYPE html><html lang="en">{% load static %}

<head><meta charset="UTF-8">

<title>Login</title>

<link rel="stylesheet"

href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css">

<style>body{background:#f5f5f5;display:flex;justify-content:center;align-items:center;min-height:100vh}.card{width:360px}</style>

</head>

<body>

<div class="card shadow"><div class="card-body">

<h4 class="text-center mb-4">Result Portal – Login</h4>

<form method="post" novalidate>

{% csrf\_token %}

{{ form.register\_number.label\_tag }} {{ form.register\_number }}

{{ form.dob.label\_tag }} {{ form.dob }}

<button class="btn btn-primary w-100 mt-3">Login</button>

{% if form.non\_field\_errors %}

<div class="text-danger small mt-2">{{ form.non\_field\_errors }}</div>

{% endif %}

</form>

<hr>

<p class="mb-1"><strong>Users Trying:</strong> {{ users\_trying }}</p>

<p class="mb-1"><strong>Routed to:</strong> {{ routed\_to }}</p>

<div class="progress mb-2" style="height:8px">

<div class="progress-bar bg-success" style="width: {{ server\_load }}%"></div>

</div>

<p class="mb-0"><strong>Server Load:</strong> {{ server\_load }}%</p>

<p class="mb-0"><strong>Status:</strong> {{ status }}</p>

</div></div>

</body></html>

**result.html**

html

CopyEdit

<!DOCTYPE html><html lang="en">

<head><meta charset="UTF-8">

<title>{{ student.register\_number }} Result</title>

<link rel="stylesheet"

href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css">

</head>

<body class="bg-light p-5">

<div class="container">

<div class="card mx-auto shadow" style="max-width:480px">

<div class="card-body text-center">

<h4>Result Sheet</h4>

<p><strong>Reg No:</strong> {{ student.register\_number }}</p>

<p><strong>Semester:</strong> {{ result.semester }}</p>

<p><strong>GPA:</strong> {{ result.gpa }}</p>

<p class="text-muted small">Updated: {{ result.updated|date:"d M Y H:i" }}</p>

<a href="{% url 'portal:login' %}" class="btn btn-outline-secondary btn-sm">Logout</a>

</div>

</div>

</div>

</body></html>

**jam\_dashboard.html**

html

CopyEdit

{% extends "base.html" %}

{% block title %}Jam Dashboard{% endblock %}

{% block content %}

<div class="container" style="max-width:400px">

<div class="card shadow p-4 mt-4">

<h4 class="text-center mb-3">Live Traffic – Jamming</h4>

<p class="mb-1"><strong>Users Trying:</strong> <span id="users">{{ users\_trying }}</span></p>

<p class="mb-1"><strong>Routed to:</strong> <span id="route">{{ routed\_to }}</span></p>

<div class="progress mb-2" style="height:8px">

<div id="bar" class="progress-bar bg-success" style="width: {{ server\_load }}%"></div>

</div>

<p class="mb-0"><strong>Server Load:</strong> <span id="load">{{ server\_load }}</span>%</p>

<p class="mb-0"><strong>Status:</strong> <span id="status">{{ status }}</span></p>

</div>

</div>

<script>

setInterval(()=>fetch("{% url 'portal:jam\_dashboard' %}?ajax=1")

.then(r=>r.json()).then(d=>{

users.textContent = d.users\_trying;

route.textContent = d.routed\_to;

load.textContent = d.server\_load;

status.textContent= d.status;

bar.style.width = d.server\_load+"%";

} ),3000);

</script>

{% endblock %}

**student\_list.html / student\_form.html / student\_confirm\_delete.html / result\_list.html / result\_form.html / result\_confirm\_delete.html** – use the versions provided in the earlier answer.

## 6️⃣ portal/management/commands/generate\_dummy\_data.py

python

CopyEdit

from django.core.management.base import BaseCommand

from faker import Faker

from portal.models import Student, Result

import random

class Command(BaseCommand):

help = "Create 200 unique fake students & results"

def handle(self, \*args, \*\*kwargs):

fake = Faker()

Student.objects.all().delete()

Result.objects.all().delete()

regs = set()

while len(regs) < 200:

reg = f"22CS{random.randint(1000,1999)}"

if reg in regs: continue

regs.add(reg)

dob = fake.date\_of\_birth(minimum\_age=17, maximum\_age=22)

s = Student.objects.create(register\_number=reg, dob=dob)

Result.objects.create(student=s,

semester=random.randint(1,8),

gpa=round(random.uniform(6.0,10.0),2))

self.stdout.write(self.style.SUCCESS("Dummy data ready!"))

(ensure *portal/management/\_\_init\_\_.py* and *portal/management/commands/\_\_init\_\_.py* are empty files)

## 7️⃣ settings.py tweaks

python

CopyEdit

INSTALLED\_APPS += ["portal"]

# Quick in‑memory cache (no Redis needed for demo)

CACHES = {

"default": {

"BACKEND": "django.core.cache.backends.locmem.LocMemCache",

"LOCATION": "jam-demo",

"TIMEOUT": 60,

}

}

### 🚀 Run

bash

CopyEdit

python manage.py makemigrations

python manage.py migrate

python manage.py generate\_dummy\_data

python manage.py runserver

* http://127.0.0.1:8000/ – login simulation
* /jam/ – live jamming dashboard
* /students/, /results/ – full CRUD

You now have the **complete Student Result Portal with jamming simulation and CRUD**.  
Happy coding! 🎉

##### You said:

remainter html code show me

###### ChatGPT said:

Thought for 4 seconds

Below are **all the remaining HTML templates** you’ll need inside  
portal/templates/portal/. Each file already extends base.html, so you only have to copy‑paste them into the correct paths.

**Tip:** keep filenames exactly as shown.

### 1️⃣ student\_list.html

html

CopyEdit

{% extends "base.html" %}

{% block title %}Students{% endblock %}

{% block content %}

<div class="container">

<h3 class="mb-3">Students</h3>

<a href="{% url 'portal:student\_add' %}" class="btn btn-sm btn-primary mb-2">+ Add Student</a>

<table class="table table-bordered table-sm">

<tr><th>Reg No</th><th>DOB</th><th></th></tr>

{% for s in object\_list %}

<tr>

<td>{{ s.register\_number }}</td>

<td>{{ s.dob }}</td>

<td>

<a href="{% url 'portal:student\_edit' s.pk %}">Edit</a> |

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

</td>

</tr>

{% empty %}

<tr><td colspan="3">No records found.</td></tr>

{% endfor %}

</table>

{% if is\_paginated %}

<nav>

<ul class="pagination">

{% if page\_obj.has\_previous %}

<li class="page-item">

<a class="page-link" href="?page={{ page\_obj.previous\_page\_number }}">&laquo;</a>

</li>

{% endif %}

{% for p in paginator.page\_range %}

<li class="page-item {% if page\_obj.number == p %}active{% endif %}">

<a class="page-link" href="?page={{ p }}">{{ p }}</a>

</li>

{% endfor %}

{% if page\_obj.has\_next %}

<li class="page-item">

<a class="page-link" href="?page={{ page\_obj.next\_page\_number }}">&raquo;</a>

</li>

{% endif %}

</ul>

</nav>

{% endif %}

</div>

{% endblock %}

### 2️⃣ student\_form.html  (create & edit)

html

CopyEdit

{% extends "base.html" %}

{% block title %}{{ view.object|yesno:"Edit Student,Add Student" }}{% endblock %}

{% block content %}

<div class="container" style="max-width:480px;">

<h4 class="mb-3">{{ view.object|yesno:"Edit Student,Add Student" }}</h4>

<form method="post" class="card p-4 shadow-sm">

{% csrf\_token %}

{{ form.as\_p }}

<button class="btn btn-primary">Save</button>

<a href="{% url 'portal:student\_list' %}" class="btn btn-secondary ms-2">Cancel</a>

</form>

</div>

{% endblock %}

### 3️⃣ student\_confirm\_delete.html

html

CopyEdit

{% extends "base.html" %}

{% block title %}Delete Student{% endblock %}

{% block content %}

<div class="container" style="max-width:480px;">

<div class="alert alert-warning shadow-sm mt-4">

<h5 class="alert-heading">Confirm delete</h5>

<p>Are you sure you want to delete <strong>{{ object.register\_number }}</strong>?</p>

<form method="post">

{% csrf\_token %}

<button class="btn btn-danger">Yes, delete</button>

<a href="{% url 'portal:student\_list' %}" class="btn btn-secondary ms-2">Cancel</a>

</form>

</div>

</div>

{% endblock %}

### 4️⃣ result\_list.html

html

CopyEdit

{% extends "base.html" %}

{% block title %}Results{% endblock %}

{% block content %}

<div class="container">

<h3 class="mb-3">Results</h3>

<a href="{% url 'portal:result\_add' %}" class="btn btn-sm btn-primary mb-2">+ Add Result</a>

<table class="table table-bordered table-sm">

<tr><th>Reg No</th><th>Semester</th><th>GPA</th><th></th></tr>

{% for r in object\_list %}

<tr>

<td>{{ r.student.register\_number }}</td>

<td>{{ r.semester }}</td>

<td>{{ r.gpa }}</td>

<td>

<a href="{% url 'portal:result\_edit' r.pk %}">Edit</a> |

<a href="{% url 'portal:result\_delete' r.pk %}">Delete</a>

</td>

</tr>

{% empty %}

<tr><td colspan="4">No results yet.</td></tr>

{% endfor %}

</table>

{% if is\_paginated %}

<nav>

<ul class="pagination">

{% if page\_obj.has\_previous %}

<li class="page-item">

<a class="page-link" href="?page={{ page\_obj.previous\_page\_number }}">&laquo;</a>

</li>

{% endif %}

{% for p in paginator.page\_range %}

<li class="page-item {% if page\_obj.number == p %}active{% endif %}">

<a class="page-link" href="?page={{ p }}">{{ p }}</a>

</li>

{% endfor %}

{% if page\_obj.has\_next %}

<li class="page-item">

<a class="page-link" href="?page={{ page\_obj.next\_page\_number }}">&raquo;</a>

</li>

{% endif %}

</ul>

</nav>

{% endif %}

</div>

{% endblock %}

### 5️⃣ result\_form.html

html

CopyEdit

{% extends "base.html" %}

{% block title %}{{ view.object|yesno:"Edit Result,Add Result" }}{% endblock %}

{% block content %}

<div class="container" style="max-width:480px;">

<h4 class="mb-3">{{ view.object|yesno:"Edit Result,Add Result" }}</h4>

<form method="post" class="card p-4 shadow-sm">

{% csrf\_token %}

{{ form.as\_p }}

<button class="btn btn-primary">Save</button>

<a href="{% url 'portal:result\_list' %}" class="btn btn-secondary ms-2">Cancel</a>

</form>

</div>

{% endblock %}

### 6️⃣ result\_confirm\_delete.html

html

CopyEdit

{% extends "base.html" %}

{% block title %}Delete Result{% endblock %}

{% block content %}

<div class="container" style="max-width:480px;">

<div class="alert alert-warning shadow-sm mt-4">

<h5 class="alert-heading">Confirm delete</h5>

<p>

Remove result for

<strong>{{ object.student.register\_number }}</strong>

(SEM {{ object.semester }})?

</p>

<form method="post">

{% csrf\_token %}

<button class="btn btn-danger">Yes, delete</button>

<a href="{% url 'portal:result\_list' %}" class="btn btn-secondary ms-2">Cancel</a>

</form>

</div>

</div>

{% endblock %}

With these templates in place, **every CRUD page, login simulation, and jamming dashboard will render correctly**.

Run python manage.py runserver, visit:

* /students/ → create, edit, delete students
* /results/ → manage results
* /jam/ → see real‑time load simulation

If you hit any other missing‑file errors, just tell me which file name Django can’t find, and I’ll fill it in!

##### You said:

nee kututha code na Student teach pannum nee step by stepe nee ennku sollu

###### ChatGPT said:

Thought for a few seconds

### 📚 Step‑by‑Step Lesson Plan – “Student Result + Jamming Portal”

(Tamil + English mix so students grasp both code & concepts)

## 🎯 ****Learning Goals****

1. Build a Django project end‑to‑end (models → views → templates).
2. Understand CRUD operations with ListView / CreateView / UpdateView / DeleteView.
3. See live traffic simulation (users‑trying, cache, server‑load) and learn basic caching.
4. Seed dummy data & test real‑world load.

## 🗓️ ****5‑Day Classroom Road‑map****

| **Day** | **Topics & Activities** | **Homework / Lab** |
| --- | --- | --- |
| **1** | • Python + virtual‑env setup • django-admin startproject result\_portal . • Create app portal & add to INSTALLED\_APPS | Install VS Code, run Hello Django page |
| **2** | • **Models** Student, Result – explain OneToOneField, unique=True • makemigrations vs migrate | Draw ER diagram; add one student via Django admin |
| **3** | • **Forms**: plain LoginForm vs ModelForm • **Views**: function‑based login/result & stats helper • URL routing basics | Modify login page style (Bootstrap colors) |
| **4** | • **Class‑based CRUD views** – L/C/U/D for Student & Result • **Templates** inheritance (base.html) & pagination • Navbar links | Add search bar to student list |
| **5** | • Live **Jam Dashboard** – JSON AJAX polling, progress‑bar update • Caching (LocMemCache / Redis), why & how • **generate\_dummy\_data** command; run hey or locust for load | Short quiz + deploy to Heroku / Render (optional) |

## 🛠️ ****Detailed Walk‑through****

### 1️⃣ Environment & Project

bash

CopyEdit

python -m venv .venv

.\.venv\Scripts\activate

pip install django faker

django-admin startproject result\_portal .

python manage.py startapp portal

Explain: why we use virtual‑env, project vs app folders.

### 2️⃣ Models (portal/models.py)

python

CopyEdit

class Student(models.Model):

register\_number = models.CharField(max\_length=20, unique=True)

dob = models.DateField()

class Result(models.Model):

student = models.OneToOneField(Student, on\_delete=models.CASCADE)

semester = models.PositiveSmallIntegerField()

gpa = models.DecimalField(max\_digits=4, decimal\_places=2)

Teach:

* OneToOneField similar to “PK–FK (1‑1)” in DB.
* unique=True prevents duplicates → shows IntegrityError if violated.

Run:

bash

CopyEdit

python manage.py makemigrations

python manage.py migrate

### 3️⃣ Forms

python

CopyEdit

class LoginForm(forms.Form): # simple form, not tied to DB

class StudentForm(ModelForm): # CRUD – auto build fields

class ResultForm(ModelForm):

Point out: Difference between plain Form (manual fields) vs ModelForm (auto‑sync with model).

### 4️⃣ Views

#### 🔹 Traffic simulator helper

python

CopyEdit

def \_simulate\_stats():

users = cache.get\_or\_set("users\_trying", 0, timeout=10)

cache.incr("users\_trying")

load = random.randint(15,90)

return {...}

Explain cache counters, why timeout=10 auto‑expires (avoids infinite growth).

#### 🔹 Login flow

1. POST → validate LoginForm.
2. Try cache for student key.
3. If DB hit → store session sid.
4. Redirect to result page.

Demo: print cache\_hit in console; watch effect after second login.

#### 🔹 Class‑based CRUD

Show code for StudentList, StudentCreate, etc.  
Explain generic view attributes (model, template\_name, success\_url).

### 5️⃣ Templates (Bootstrap)

Key ideas:

* {% extends "base.html" %} → DRY layout.
* Pagination block uses page\_obj.
* jam\_dashboard.html uses JS fetch every 3 s → live update.

Classroom demo: change progress‑bar color by adding:

html

CopyEdit

{% if server\_load >= 50 %}

bar.classList.remove("bg-success"); bar.classList.add("bg-danger");

{% endif %}

### 6️⃣ URL Routing

Walk through:

python

CopyEdit

path("", views.login\_view) # root

path("jam/", views.jam\_dashboard) # live stats

path("students/", StudentList.as\_view()) # CRUD group

Explain app\_name = "portal" and reverse lookup {% url 'portal:student\_add' %}.

### 7️⃣ Seeding Data & Load Test

bash

CopyEdit

python manage.py generate\_dummy\_data

hey -n 500 -c 50 http://127.0.0.1:8000/

Students observe Jam Dashboard spiking.

Concepts: concurrency (-c 50), total requests (-n 500).

## 🎓 ****Evaluation****

* Mini‑quiz: “Why do we cache login lookups?”
* Hands‑on: Add search query param to StudentList (filter register\_number\_\_icontains).
* Stretch: deploy to Render; configure Postgres + Redis.

## 🗣️ ****Tamil Recap (Very Short)****

1. **Day 1:** Django install, project create.
2. **Day 2:** Models – Student & Result.
3. **Day 3:** Login form + views, template inheritance.
4. **Day 4:** CRUD generic views, pagination.
5. **Day 5:** Jam dashboard, cache, load test.

இப்படிப் ப்ளான் பண்ணி நீங்க class‑ல் live‑ஆ demonstrate பண்ணலாம். Doubt இருந்தா next கேள் – code tweak பண்ணிப் தரேன்! 👍

##### You said:

entha project la code logic ennun ketta na ippati solluranthu

###### ChatGPT said:

### 🔍 Project‑ஆலோசனை (Code Logic) – “Student Result + Jamming Portal”

(உங்க நண்பன்/மாணவரு  “***Code logic enna?***” னு கேட்டா, இப்படி சிம்பிளா சொல்லலாம்)

## 1. Data Layer (Models)

| **Model** | **Purpose** | **Key Fields** |
| --- | --- | --- |
| **Student** | ஒவ்வொரு மாணவரும் unique Reg‑No + DOB வைத்திருக்கிறார் | register\_number (unique), dob |
| **Result** | ஒவ்வொரு Student‑க்கு ஒரு Semester GPA | student (OneToOne), semester, gpa |

**Logic:** 1‑to‑1 relationship → Student record அழிந்தால், Result auto delete (Cascade).

## 2. Login Flow (Function‑based View)

txt

CopyEdit

POST / (Login Form submit)

│

├─ Validate Reg‑No + DOB ✅/❌

├─ Try Redis/LocMem cache → student? (Hit / Miss)

│ └─ If miss → query DB, cache for 60s

└─ Success → save student.id in session → redirect /home/

**Logic:** Cache first → database second → speed & lower load.

## 3. Result Page

sql

CopyEdit

/home/

└─ session["sid"] → fetch Student + Result → render result.html

If session missing, redirect back to login.

## 4. Traffic Simulation (\_simulate\_stats())

python

CopyEdit

users = cache.get\_or\_set("users\_trying", 0, timeout=10)

cache.incr("users\_trying") # auto‑resets every 10 s

server\_load = random 15‑90 %

status = "High Load" if load ≥ 50 else "Normal"

**Logic:** Every request bumps a short‑lived counter → dashboard shows “Users Trying”.  
Random load mimics real server strain.

## 5. Jam Dashboard (/jam/)

| **Action** | **Output** |
| --- | --- |
| Normal GET | Render card with current stats |
| GET ?ajax=1 | Return JSON {users\_trying, routed\_to, server\_load, status} |

Frontend JS polls /jam/?ajax=1 every 3 s → updates DOM.

## 6. CRUD Layer (Class‑based Views)

| **View** | **Logic** |
| --- | --- |
| StudentList / ResultList | ListView + pagination |
| StudentCreate / ResultCreate | CreateView + ModelForm |
| Update, Delete | Same pattern → |

**Benefit:** Generic views reduce boilerplate; only set model, form\_class, success\_url.

## 7. Dummy Data Command

python

CopyEdit

while len(regs)<200:

reg = f"22CS{randint(1000,1999)}" (ensure unique)

create Student + Result

**Logic:** Generates sample records without duplicates → useful for demo & load test.

## 8. Caching Strategy

* **LocMemCache (demo) / Redis (prod)**
* Keys: "users\_trying" counter, "student:<reg>" object cache
* Short TTL → data stays fresh.

## 9. Why “Jamming” Matters

* Shows students live **impact of concurrent traffic**.
* Teaches **cache hits reduce DB queries**.
* Visual progress‑bar = instant feedback during load testing (hey, locust).

### 🗣️ Tamil “Short Answer”

“இந்த project‑ல logic simple:

1. Student & Result table;
2. Login பண்ணப்போ cache‑ல first check, miss ஆனா DB;
3. Session‑ல id save பண்ணி result காட்டுறோம்;
4. ஒவ்வொரு request‑க்கும் ‘users\_trying’, random server‑load generate பண்ணி Jam Dashboard‑ல live‑ஆ update பண்ணப் போறோம்;
5. CRUD generic views‑உம் dummy‑data generator‑உம் கொண்டு full admin control கிடைக்குது.”

இதுக்குப் பின்னால வேற doubt இருந்தா கேளுங்க! 👍

##### You said:

{% extends "portal/base.html" %} {% block title %}{{ view.object|yesno:"Edit Result,Add Result" }}{% endblock %} {% block content %} <div class="container" style="max-width:480px;"> <h4 class="mb-3">{{ view.object|yesno:"Edit Result,Add Result" }}</h4> <form method="post" class="card p-4 shadow-sm"> {% csrf\_token %} {{ form.as\_p }} <button class="btn btn-primary">Save</button> <a href="{% url 'portal:result\_list' %}" class="btn btn-secondary ms-2">Cancel</a> </form> </div> {% endblock %} student add pannu dilaa corec

###### ChatGPT said:

Thought for 5 seconds

### ✅ Correct “Student Form” template

(use this for **Add Student** & **Edit Student** pages)

Save as **portal/templates/portal/student\_form.html**

html

CopyEdit

{% extends "base.html" %}

{% block title %}

{% if view.object %}Edit Student{% else %}Add Student{% endif %}

{% endblock %}

{% block content %}

<div class="container" style="max-width:480px;">

<h4 class="mb-3">

{% if view.object %}Edit Student{% else %}Add Student{% endif %}

</h4>

<form method="post" class="card p-4 shadow-sm">

{% csrf\_token %}

{{ form.as\_p }}

<button class="btn btn-primary">Save</button>

<a href="{% url 'portal:student\_list' %}" class="btn btn-secondary ms-2">Cancel</a>

</form>

</div>

{% endblock %}

#### 🔑 Fixes made

1. **Extends**  
   {% extends "base.html" %}  
   (no portal/ prefix; Django already looks inside the app’s templates folder).
2. **Heading & title logic**  
   Replaced the yesno filter with a plain {% if view.object %} block—simpler and 100 % reliable:
   * When **creating**, view.object is None → shows **“Add Student”**.
   * When **editing**, view.object is the existing Student instance → shows **“Edit Student”**.
3. **Path back to list**  
   Cancel button links to {% url 'portal:student\_list' %}.

இப்போ /students/add/ গেলে “Add Student” form வரும், /students/123/edit/ গেলে “Edit Student” form வரும்— headings & browser title சரியாகப் போட்டுத்தரும்.  
ஏதாவது rendering error அல்லது path issue இருந்தா message copy‑paste பண்ணுங்க; next fix பண்ணி தரேன்! 👍

**frontend** எல்லாமே முழுமையாக படிப்படியாக **தமிழில் (with English terms)** teaching பண்ண்றேன். இது beginner-friendly guide.

## 🎓 Student CRUD REST API Project — Teaching in Tamil

### 📌 1. என்ன project?

நம்ம இப்போ Student resource-க்கு:

| **Operation** | **Description** |
| --- | --- |
| Create | Student-ஐ add பண்ணுவோம் |
| Read | Student-ஐ list பண்ணுவோம் |
| Update | Student details-ஐ edit பண்ணலாம் |
| Delete | Student-ஐ remove பண்ணலாம் |

## 🧱 2. Backend (Django REST Framework)

### ✅ Step 1: Model Definition (models.py)

python

CopyEdit

from django.db import models

class Student(models.Model):

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

age = models.IntegerField()

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

def \_\_str\_\_(self):

return self.name

இது Student table-க்கு base. ஒவ்வொரு student-க்கும் name, age, course இருக்குமா என define பண்ணிருக்கோம்.

### ✅ Step 2: Serializer (serializers.py)

python

CopyEdit

from rest\_framework import serializers

from .models import Student

class StudentSerializer(serializers.ModelSerializer):

class Meta:

model = Student

fields = '\_\_all\_\_'

**Serializers**னா model data-ஐ JSON-a மாத்தும் வேலை.

### ✅ Step 3: Views (CRUD API) – views.py

python

CopyEdit

from rest\_framework import generics

from .models import Student

from .serializers import StudentSerializer

# GET + POST

class StudentListCreate(generics.ListCreateAPIView):

queryset = Student.objects.all()

serializer\_class = StudentSerializer

# GET + PUT + DELETE

class StudentDetail(generics.RetrieveUpdateDestroyAPIView):

queryset = Student.objects.all()

serializer\_class = StudentSerializer

### ✅ Step 4: URLs (urls.py)

python

CopyEdit

from django.urls import path

from .views import StudentListCreate, StudentDetail

urlpatterns = [

path('students/', StudentListCreate.as\_view()),

path('students/<int:pk>/', StudentDetail.as\_view()),

]

### ✅ Step 5: Run Migrations

bash

CopyEdit

python manage.py makemigrations

python manage.py migrate

இது தான் உங்கள் error-க்கு solution: "no such table: studentapp\_student" → means migration பண்ணல.

## 🌐 3. Frontend — HTML + JavaScript

Create a file called index.html:

html

CopyEdit

<!DOCTYPE html>

<html>

<head><title>Student CRUD</title></head>

<body>

<h2>Add Student</h2>

<form id="form">

<input id="name" placeholder="Name" required>

<input id="age" type="number" placeholder="Age" required>

<input id="course" placeholder="Course" required>

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

</form>

<div id="list"></div>

<script>

const API = 'http://127.0.0.1:8000/api/students/';

async function loadStudents() {

const res = await fetch(API);

const data = await res.json();

document.getElementById("list").innerHTML = data.map(s => `

<div>

${s.name} | ${s.age} | ${s.course}

<button onclick="deleteStudent(${s.id})">Delete</button>

</div>

`).join('');

}

document.getElementById("form").onsubmit = async function(e) {

e.preventDefault();

await fetch(API, {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({

name: document.getElementById("name").value,

age: document.getElementById("age").value,

course: document.getElementById("course").value

})

});

loadStudents();

}

async function deleteStudent(id) {

await fetch(API + id + '/', { method: 'DELETE' });

loadStudents();

}

loadStudents();

</script>

</body>

</html>

## ✅ Final Recap

| **Step** | **Description** |
| --- | --- |
| 1. | Student model create பண்ணணும் |
| 2. | Serializer create பண்ணணும் |
| 3. | Views – CRUD REST API use பண்ணணும் |
| 4. | URLs register பண்ணணும் |
| 5. | makemigrations, migrate பண்ணணும் |
| 6. | HTML + JS frontend fetch API use பண்ணணும் |

## ❓Next: Want me to...

* 🔐 Add Login system?
* ⚛️ Use React frontend instead of HTML?
* 📦 Send zip file full project?

Let me know! I can teach each part deeply.

Ask ChatGPT

##### You said:

reast appi nan

###### ChatGPT said:

நீங்க சொன்னது **"reast appi nan"** → இது **"REST API என்ன?"** (i.e., What is REST API?) என்று கேட்கிறீர்களா?

அப்படின்னா, கீழே explanation:

## 🔍 REST API என்றால் என்ன?

**REST API** (Representational State Transfer - Application Programming Interface) என்பது:

👉 Web application-களில் **data-ஐ send/receive பண்ண** ஒரு விதமான **communication method**.

## 📦 REST API யை எதற்காக use பண்ணுறோம்?

* **Frontend** (React / HTML / Android app)
* ↔️ interact பண்ண
* **Backend** (Django / Node / Java Spring) உடன்

🔁 JSON format-ல் data fetch, create, update, delete பண்ண.

## 🔑 REST API Basic Operations (CRUD)

| **Operation** | **HTTP Method** | **URL Example** | **Description** |
| --- | --- | --- | --- |
| Create | POST | /api/students/ | புதிய student add |
| Read All | GET | /api/students/ | எல்லா students-ஐ பெற |
| Read One | GET | /api/students/1/ | ஒரு student details |
| Update | PUT | /api/students/1/ | student update பண்ண |
| Delete | DELETE | /api/students/1/ | student delete பண்ண |

## 🏗 REST API Example (JSON Data)

json

CopyEdit

{

"name": "Ajith",

"age": 22,

"course": "Full Stack Developer"

}

## 🧠 Simple Understanding

🧍‍♂️User clicks "Add Student" →  
📱 Frontend: JS fetch() →  
🌐 API call (POST /api/students/) →  
🖥 Backend: Django creates new record in DB

## 🔧 Example Django URL

bash

CopyEdit

GET http://127.0.0.1:8000/api/students/

Response:

json

CopyEdit

[

{ "id": 1, "name": "Ajith", "age": 22, "course": "Python" },

{ "id": 2, "name": "Kumar", "age": 23, "course": "Java" }

]

## ✅ Summary

* **REST API** = Rules to access server data via HTTP
* Used in web/mobile apps to talk with backend
* **CRUD** operations map to GET, POST, PUT, DELETE

📌 Do you want a **Tamil video**, **React version**, or **JWT login with REST API** also?

Let me know 🙌