forms.py  
“from django import forms

from django.contrib.auth.forms import PasswordChangeForm

from .models import User, Record

class DocumentUploadForm(forms.ModelForm):

class Meta:

model = Record

fields = ['title', 'department', 'document'] # Ensure these match the model fields

class ProfileUpdateForm(forms.ModelForm):

class Meta:

model = User

fields = ['first\_name', 'last\_name', 'email', 'profile\_picture', 'department']

widgets = {

'first\_name': forms.TextInput(attrs={'class': 'form-control'}),

'last\_name': forms.TextInput(attrs={'class': 'form-control'}),

'email': forms.EmailInput(attrs={'class': 'form-control'}),

'profile\_picture': forms.FileInput(attrs={'class': 'form-control'}),

'department': forms.Select(attrs={'class': 'form-control'}),

}

class CustomPasswordChangeForm(PasswordChangeForm):

def \_\_init\_\_(self, \*args, \*\*kwargs):

super(CustomPasswordChangeForm, self).\_\_init\_\_(\*args, \*\*kwargs)

# Customize form fields here if needed

for field in self.fields.values():

field.widget.attrs['class'] = 'form-control'

”

urls.py

“from django.contrib import admin

from django.urls import path

from records import views

from . import settings

from django.conf.urls.static import static

from django.urls import include

urlpatterns = [

path('admin/', admin.site.urls),

path('', views.index, name='index'),

path('accounts/login/', views.login\_view, name='login'), # Corrected path for login

path('register/', views.register\_view, name='register'),

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

path('dashboard/', views.dashboard\_view, name='dashboard'),

path('records/', views.records\_view, name='records'),

path('departments/', views.departments\_view, name='departments'),

path('profile/', views.profile\_view, name='profile'),

path('departments/<int:department\_id>/', views.department\_records\_view, name='department\_records'),

path('departments/<int:department\_id>/upload/', views.upload\_document, name='upload\_document'),

path('departments/<int:department\_id>/', views.department\_records, name='department\_records'),

path('departments/<int:department\_id>/upload/', views.upload\_document, name='upload\_document'),

path('document/view/<int:record\_id>/', views.view\_document, name='view\_document'),

path('document/download/<int:record\_id>/', views.download\_document, name='download\_document'),

path('document/view/<int:record\_id>/', views.view\_document, name='view\_document'),

] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

”

models.py  
“# models.py

from django.db import models

from django.contrib.auth.models import AbstractUser

from django.conf import settings

class User(AbstractUser):

department = models.ForeignKey('Department', on\_delete=models.CASCADE, null=True, blank=True)

is\_admin = models.BooleanField(default=False)

profile\_picture = models.ImageField(upload\_to='profile\_pics/', default='default.jpg', blank=True, null=True)

groups = models.ManyToManyField(

'auth.Group',

related\_name='custom\_user\_set', # Unique related name

blank=True,

help\_text='The groups this user belongs to. A user will get all permissions granted to each of their groups.',

verbose\_name='groups'

)

user\_permissions = models.ManyToManyField(

'auth.Permission',

related\_name='custom\_user\_permissions\_set', # Unique related name

blank=True,

help\_text='Specific permissions for this user.',

verbose\_name='user permissions'

)

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

return self.username

class Department(models.Model):

name = models.CharField(max\_length=255, unique=True, db\_index=True)

description = models.TextField(blank=True, null=True)

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

return self.name

class Record(models.Model):

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

department = models.ForeignKey(Department, on\_delete=models.CASCADE, related\_name='records')

created\_by = models.ForeignKey(settings.AUTH\_USER\_MODEL, on\_delete=models.SET\_NULL, null=True, blank=True)

created\_at = models.DateTimeField(auto\_now\_add=True)

document = models.FileField(upload\_to='documents/')

uploaded\_at = models.DateTimeField(auto\_now\_add=True)

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

return self.title

class UserProfile(models.Model):

user = models.OneToOneField(settings.AUTH\_USER\_MODEL, on\_delete=models.CASCADE)

profile\_picture = models.ImageField(upload\_to='profile\_pictures/', null=True, blank=True)

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

return f"{self.user.username}'s Profile"

”

views.py

“from django.http import HttpResponse

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

from django.conf import settings

from .forms import DocumentUploadForm, ProfileUpdateForm, CustomPasswordChangeForm

from .models import Department, Record, UserProfile

from django.contrib.auth import authenticate, login, logout, update\_session\_auth\_hash

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

from django.contrib.auth.models import User

from django.views.generic import TemplateView

from django.db.models.signals import post\_save

from django.dispatch import receiver

from django.contrib import messages

import logging

from django.db.models import Q

from django.http import FileResponse

from .models import Record

# Initialize the logger

logger = logging.getLogger(\_\_name\_\_)

# Index view

class IndexView(TemplateView):

template\_name = 'index.html'

# Login view

def login\_view(request):

if request.method == 'POST':

username = request.POST['username']

password = request.POST['password']

user = authenticate(request, username=username, password=password)

if user is not None:

login(request, user)

return redirect('dashboard') # Redirect to the user dashboard

else:

messages.error(request, 'Invalid username or password.')

return render(request, 'accounts/login.html')

# Registration view

def register\_view(request):

if request.method == 'POST':

username = request.POST['username']

password = request.POST['password']

email = request.POST['email']

try:

user = User.objects.create\_user(username, email, password)

user.save()

messages.success(request, 'Account created successfully!')

return redirect('login')

except:

messages.error(request, 'Username already exists.')

return render(request, 'accounts/register.html')

# Logout view

def logout\_view(request):

logout(request)

return redirect('login')

@login\_required

def dashboard\_view(request):

logger.debug(f"Attempting to render dashboard for user: {request.user}")

total\_departments = Department.objects.count()

total\_records = Record.objects.count()

recent\_activities = Record.objects.order\_by('-created\_at')[:5] # Example: Get the 5 most recent records

context = {

'total\_departments': total\_departments,

'total\_records': total\_records,

'recent\_activities': recent\_activities,

}

return render(request, 'dashboard.html', context)

@login\_required

def records\_view(request):

return render(request, 'records.html')

@login\_required

def departments\_view(request):

departments = Department.objects.all()

return render(request, 'departments.html', {'departments': departments})

@login\_required

def department\_records\_view(request, department\_id):

department = get\_object\_or\_404(Department, id=department\_id)

records = department.records.all() # Use the custom related name

return render(request, 'department\_records.html', {'department': department, 'records': records})

@login\_required

def upload\_document(request, department\_id):

department = get\_object\_or\_404(Department, id=department\_id)

if request.method == 'POST':

form = DocumentUploadForm(request.POST, request.FILES)

secret\_key = request.POST.get('secret\_key')

# Check secret key before uploading

if secret\_key == department.secret\_key:

if form.is\_valid():

record = form.save(commit=False)

record.department = department

record.save()

return redirect('department\_records', department\_id=department.id)

else:

messages.error(request, "Invalid form submission.")

else:

messages.error(request, "Invalid secret key.")

else:

form = DocumentUploadForm()

context = {'form': form, 'department': department}

return render(request, 'upload\_document.html', context)

@login\_required

def profile\_view(request):

if request.method == 'POST':

# Handle profile update

profile\_form = ProfileUpdateForm(request.POST, request.FILES, instance=request.user.userprofile)

# Handle password change

password\_form = CustomPasswordChangeForm(user=request.user, data=request.POST)

if profile\_form.is\_valid():

profile\_form.save()

messages.success(request, 'Your profile picture has been updated successfully!')

if password\_form.is\_valid():

password\_form.save()

update\_session\_auth\_hash(request, password\_form.user) # Keeps the user logged in after password change

messages.success(request, 'Your password has been changed successfully!')

if profile\_form.is\_valid() or password\_form.is\_valid():

return redirect('profile') # Redirect to the profile page

else:

messages.error(request, 'Please correct the errors below.')

else:

profile\_form = ProfileUpdateForm(instance=request.user.userprofile)

password\_form = CustomPasswordChangeForm(user=request.user)

context = {

'profile\_form': profile\_form,

'password\_form': password\_form

}

return render(request, 'profile.html', context)

# Signal to create or update user profile

@receiver(post\_save, sender=settings.AUTH\_USER\_MODEL)

def create\_or\_update\_user\_profile(sender, instance, created, \*\*kwargs):

if created:

UserProfile.objects.create(user=instance)

else:

instance.userprofile.save()

# Searching for department

def department\_list(request):

query = request.GET.get('q') # Get the search query from the request

if query:

departments = Department.objects.filter(name\_\_icontains=query) # Case-insensitive search

else:

departments = Department.objects.all() # Show all departments if no search query

context = {

'departments': departments,

'query': query,

}

return render(request, 'departments.html', context)

# Index view

def index(request):

return render(request, 'index.html')

@login\_required

def records\_view(request):

records = Record.objects.all().select\_related('department', 'created\_by') # Fetch all records with related department and user

return render(request, 'records.html', {'records': records})

@login\_required

def department\_records(request, department\_id):

department = get\_object\_or\_404(Department, id=department\_id)

query = request.GET.get('q')

records = Record.objects.filter(department=department).order\_by('title') # Sort alphabetically by title

if query:

records = records.filter(title\_\_icontains=query)

return render(request, 'department\_records.html', {'department': department, 'records': records})

#Document upload

@login\_required

def upload\_document(request, department\_id):

department = get\_object\_or\_404(Department, id=department\_id)

if request.method == 'POST' and request.FILES.get('document'):

document = request.FILES['document']

title = document.name

# Create a new record

Record.objects.create(

department=department,

title=title,

document=document,

created\_by=request.user

)

return redirect('department\_records', department\_id=department\_id)

return redirect('department\_records', department\_id=department\_id)

@login\_required

def department\_records(request, department\_id):

department = get\_object\_or\_404(Department, id=department\_id)

query = request.GET.get('q', '')

records = Record.objects.filter(department=department).order\_by('title')

if query:

records = records.filter(title\_\_icontains=query)

return render(request, 'department\_records.html', {'department': department, 'records': records})

def view\_document(request, record\_id):

# Retrieve the specific record using its ID

record = get\_object\_or\_404(Record, id=record\_id)

# Open the document file and create a response

response = FileResponse(record.document.open('rb'), content\_type='application/pdf')

response['Content-Disposition'] = f'inline; filename="{record.document.name}"'

return response

# View to handle document download

@login\_required

def download\_document(request, record\_id):

# Retrieve the specific record using its ID

record = get\_object\_or\_404(Record, id=record\_id)

# Open the document file and create a response for download

response = FileResponse(record.document.open('rb'), content\_type='application/octet-stream')

response['Content-Disposition'] = f'attachment; filename="{record.document.name}"'

return response

”

Department\_records.html

“{% load static %}

<link rel="stylesheet" href="{% static 'css/department\_records.css' %}">

{% block content %}

<div class="container mt-5">

<h2 class="text-center">{{ department.name }} Records</h2>

<!-- Upload Form -->

<form method="post" enctype="multipart/form-data" class="mb-4">

{% csrf\_token %}

<input type="file" name="document" class="form-control-file mb-2">

<button type="submit" class="btn btn-primary">Upload</button>

</form>

<!-- Search Form -->

<form method="GET" action="{% url 'department\_records' department.id %}" class="search-form mb-3">

<div class="input-group">

<input type="text" name="q" class="form-control" placeholder="Search for a document..." value="{{ request.GET.q }}">

<button class="btn btn-secondary" type="submit">Search</button>

</div>

</form>

<!-- Records Table -->

<table class="table table-bordered table-striped mt-3">

<thead class="thead-dark">

<tr>

<th>Title</th>

<th>Uploaded By</th>

<th>Upload Date</th>

<th>Upload Time</th>

<th>Actions</th>

</tr>

</thead>

<tbody>

{% for record in records %}

<tr>

<td>{{ record.title }}</td>

<td>{{ record.created\_by.username }}</td>

<td>{{ record.uploaded\_at|date:"Y-m-d" }}</td>

<td>{{ record.uploaded\_at|date:"H:i:s" }}</td>

<td>

<a href="{% url 'view\_document' record.id %}" class="btn btn-info btn-sm" target="\_blank">View</a>

<a href="{% url 'download\_document' record.id %}" class="btn btn-success btn-sm">Download</a>

</td>

</tr>

{% empty %}

<tr>

<td colspan="5" class="text-center">No records found.</td>

</tr>

{% endfor %}

</tbody>

</table>

</div>

{% endblock %}

”  
  
Records.html

“{% load static %}

<link rel="stylesheet" href="{% static 'css/records.css' %}">

{% block content %}

<div class="container mt-5">

<h2 class="text-center">All Records</h2>

<table class="table table-bordered table-striped mt-4">

<thead class="thead-dark">

<tr>

<th>Department</th>

<th>Title</th>

<th>Uploaded By</th>

<th>Upload Date</th>

<th>Upload Time</th>

</tr>

</thead>

<tbody>

{% for record in records %}

<tr>

<td>{{ record.department.name }}</td>

<td>{{ record.title }}</td>

<td>{{ record.created\_by.username }}</td>

<td>{{ record.uploaded\_at|date:"Y-m-d" }}</td>

<td>{{ record.uploaded\_at|date:"H:i:s" }}</td>

</tr>

{% endfor %}

</tbody>

</table>

</div>

{% endblock %}

”

upload\_documents.html

“<h2>Upload Document for {{ department.name }}</h2>

{% if error %}

<p style="color: red;">{{ error }}</p>

{% endif %}

<form method="post" enctype="multipart/form-data">

{% csrf\_token %}

{{ form.as\_p }}

<label for="secret\_key">Secret Key:</label>

<input type="password" name="secret\_key" required>

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

</form>

”

Dashboard.html   
“{% load static %}

<link rel="stylesheet" href="{% static 'css/dashboard.css' %}">

{% block content %}

<div class="dashboard-container">

<!-- Sidebar -->

<nav class="dashboard-nav">

<ul class="user-details">

<li class="nav-item">

<a class="btn" href="{% url 'records' %}">

<i class="fas fa-file-alt"></i> Records

</a>

</li>

<li class="nav-item">

<a class="btn" href="{% url 'departments' %}">

<i class="fas fa-building"></i> Departments

</a>

</li>

<li class="nav-item">

<a class="btn" href="{% url 'profile' %}">

<i class="fas fa-user"></i> Settings

</a>

</li>

</ul>

<!-- Logout Button -->

<div class="logout-section" style="margin-top: 90%;">

<a href="{% url 'index' %}" class="btn logout-btn">

<i class="fas fa-sign-out-alt"></i> Logout

</a>

</div>

</nav>

<!-- Main Content -->

<main class="dashboard-main-content">

<div class="total-info">

<h2>Dashboard</h2>

<div class="btn-toolbar">

<button type="button" class="btn btn-small">Add New Record</button>

</div>

</div>

<!-- Summary Cards -->

<div class="dashboard-section content">

<div class="widget">

<h3>Total Records</h3>

<p>120</p>

</div>

<div class="widget">

<h3>Departments</h3>

<p>12</p>

</div>

<!-- More summary cards can be added here -->

</div>

<br>

<!-- Recent Activities or Records Table -->

<section class="recent-activities">

<h2>Recent Activities</h2>

<div class="table-responsive">

<table class="table">

<thead>

<tr>

<th>#</th>

<th>Record Name</th>

<th>Date Added</th>

<th>Department</th>

<th>Action</th>

</tr>

</thead>

<tbody>

<tr>

<td>1</td>

<td>Record A</td>

<td>2024-08-27</td>

<td>ICT</td>

<td><button class="btn btn-small">View</button></td>

</tr>

<!-- More rows can be added dynamically -->

</tbody>

</table>

</div>

</section>

</main>

</div>

{% endblock %}

”

settings.py

“"""

Django settings for records\_storage project.

Generated by 'django-admin startproject' using Django 5.1.

For more information on this file, see

https://docs.djangoproject.com/en/5.1/topics/settings/

For the full list of settings and their values, see

https://docs.djangoproject.com/en/5.1/ref/settings/

"""

import os

from pathlib import Path

# Build paths inside the project like this: BASE\_DIR / 'subdir'.

BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent

# Quick-start development settings - unsuitable for production

# See https://docs.djangoproject.com/en/5.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!

# SECURITY WARNING: don't run with debug turned on in production!

DEBUG = True

ALLOWED\_HOSTS = []

# Application definition

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'records',

'records\_storage',

]

MIDDLEWARE = [

'django.middleware.security.SecurityMiddleware',

'django.contrib.sessions.middleware.SessionMiddleware',

'django.middleware.common.CommonMiddleware',

'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',

'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

ROOT\_URLCONF = 'records\_storage.urls'

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': [BASE\_DIR / 'records\_storage' / 'records' / 'templates'],

'APP\_DIRS': True,

'OPTIONS': {

'context\_processors': [

'django.template.context\_processors.debug',

'django.template.context\_processors.request',

'django.contrib.auth.context\_processors.auth',

'django.contrib.messages.context\_processors.messages',

],

},

},

]

WSGI\_APPLICATION = 'records\_storage.wsgi.application'

# Database

# https://docs.djangoproject.com/en/5.1/ref/settings/#databases

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': 'moict\_records',

'USER': 'abel',

'PASSWORD': 'Abel123',

'HOST': 'localhost',

'PORT': '3306',

}

}

# Password validation

# https://docs.djangoproject.com/en/5.1/ref/settings/#auth-password-validators

AUTH\_PASSWORD\_VALIDATORS = [

{

'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

},

]

# Internationalization

# https://docs.djangoproject.com/en/5.1/topics/i18n/

LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'UTC'

USE\_I18N = True

USE\_TZ = True

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/5.1/howto/static-files/

STATIC\_URL = 'static/'

STATICFILES\_DIRS = [os.path.join(BASE\_DIR, 'records/static')]

# Default primary key field type

# https://docs.djangoproject.com/en/5.1/ref/settings/#default-auto-field

DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'

# settings.py

MEDIA\_URL = '/media/'

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media')

”