**Models.py**

# from django.db import models

# Create your models here.

# models.py

from django.db import models

from django.contrib.auth.models import User

from django.contrib.auth.models import AbstractUser

from django.core.validators import EmailValidator

from django.core.exceptions import ValidationError

from PIL import Image

import os

# Validator to ensure the image size does not exceed the limit

def validate\_image\_size(image):

    filesize = image.size

    megabyte\_limit = 2.0  # Maximum allowed size is 2 MB

    if filesize > megabyte\_limit \* 1024 \* 1024:

        raise ValidationError(f"Image size cannot exceed {megabyte\_limit}MB")

# Function to generate the upload path for profile pictures

def profile\_picture\_path(instance, filename):

    # Save profile pictures under: media/profile\_pictures/<user\_id>/<filename>

    return f"profile\_pictures/{instance.id}/{filename}"

# Custom User model with profile picture and additional fields

class User(AbstractUser):

    ROLE\_CHOICES = [

        ('faculty', 'Faculty'),

        ('admin', 'Admin'),

        ('external', 'External Partner'),

    ]

    email = models.EmailField(

        unique=True,

        validators=[EmailValidator()],

        error\_messages={

            'unique': 'A user with this email already exists.',

        }

    )

    role = models.CharField(

        max\_length=20,

        choices=ROLE\_CHOICES

        # default='faculty'

    )

    last\_login\_attempt = models.DateTimeField(null=True, blank=True)

    login\_attempts = models.IntegerField(default=0)

    # Profile picture field

    profile\_picture = models.ImageField(

        upload\_to=profile\_picture\_path,

        validators=[validate\_image\_size],

        null=True,

        blank=True,

        help\_text="Upload a JPEG or PNG image (max size: 2MB)."

    )

    USERNAME\_FIELD = 'email'

    REQUIRED\_FIELDS = ['username', 'role']

    class Meta:

        db\_table = 'auth\_user'

    def \_\_str\_\_(self):

        return self.email

    # Override save method to resize profile picture if necessary

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

        super().save(\*args, \*\*kwargs)

        if self.profile\_picture:

            img = Image.open(self.profile\_picture.path)

            max\_size = (300, 300)  # Resize image to a maximum of 300x300

            if img.height > max\_size[1] or img.width > max\_size[0]:

                img.thumbnail(max\_size)

                img.save(self.profile\_picture.path)

    # Override delete method to remove profile picture file when user is deleted

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

        if self.profile\_picture and os.path.isfile(self.profile\_picture.path):

            os.remove(self.profile\_picture.path)

        super().delete(\*args, \*\*kwargs)

class File(models.Model):

    user = models.ForeignKey(User, on\_delete=models.CASCADE)

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

    file = models.FileField(upload\_to='files/')

    file\_size = models.BigIntegerField()

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

    modified\_at = models.DateTimeField(auto\_now=True)

    def \_\_str\_\_(self):

        return self.name

class Collaboration(models.Model):

    user = models.ForeignKey(User, on\_delete=models.CASCADE)

    file = models.ForeignKey(File, on\_delete=models.CASCADE)

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

    def \_\_str\_\_(self):

        return f"{self.user.username} - {self.file.name}"

class Notification(models.Model):

    user = models.ForeignKey(User, on\_delete=models.CASCADE)

    message = models.TextField()

    icon = models.CharField(max\_length=50)

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

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

    def \_\_str\_\_(self):

        return f"{self.user.username} - {self.message[:50]}"

**Views.py**

from django.shortcuts import render, redirect

from django.contrib.auth import authenticate, login, logout

from django.contrib import messages

from django.contrib.auth.models import User

from django.views.decorators.csrf import csrf\_protect

from django.utils import timezone

from django.http import JsonResponse

from django.urls import reverse

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

from django.db.models import Sum

import json

from .forms import LoginForm, UserRegistrationForm

from .models import File, Collaboration, Notification

def home(request):

    return render(request, 'authentication/home.html')

@csrf\_protect

def login\_view(request):

    if request.method == 'POST':

        is\_ajax = request.headers.get('x-requested-with') == 'XMLHttpRequest'

        if is\_ajax:

            data = json.loads(request.body)

            form = LoginForm(data)

        else:

            form = LoginForm(request.POST)

        if form.is\_valid():

            email = form.cleaned\_data.get('email')

            password = form.cleaned\_data.get('password')

            role = form.cleaned\_data.get('role')

            remember\_me = form.cleaned\_data.get('remember\_me')

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

            if user is not None and user.role == role:

                if not remember\_me:

                    request.session.set\_expiry(0)

                login(request, user)

                user.last\_login = timezone.now()

                user.login\_attempts = 0

                user.save()

                if is\_ajax:

                    return JsonResponse({

                        'success': True,

                        'redirect\_url': reverse('authentication:dashboard')  # CHANGED LINE

                    })

                return redirect('authentication:dashboard')

            else:

                error\_message = 'Invalid credentials or the selected role does not match your account role.'

                if is\_ajax:

                    return JsonResponse({

                        'success': False,

                        'message': error\_message

                    }, status=400)

                messages.error(request, error\_message)

        else:

            if is\_ajax:

                return JsonResponse({

                    'success': False,

                    'message': form.errors

                }, status=400)

            messages.error(request, form.errors)

    else:

        form = LoginForm()

    return render(request, 'authentication/login.html', {'form': form})

@csrf\_protect

def register\_view(request):

    if request.method == 'POST':

        is\_ajax = request.headers.get('x-requested-with') == 'XMLHttpRequest'

        # Print received data for debugging

        print("POST data:", request.POST)

        print("FILES:", request.FILES)

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

        if not form.is\_valid():

            print("Form errors:", form.errors)

            error\_response = {

                'success': False,

                'message': {field: str(errors[0]) for field, errors in form.errors.items()}

            }

            if is\_ajax:

                return JsonResponse(error\_response, status=400)

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

            return render(request, 'authentication/register.html', {'form': form})

        try:

            user = form.save()  # This will now handle username creation

            success\_message = 'Registration successful! Please log in.'

            if is\_ajax:

                return JsonResponse({

                    'success': True,

                    'message': success\_message,

                    'redirect\_url': reverse('authentication:login')

                })

            messages.success(request, success\_message)

            return redirect('authentication:login')

        except Exception as e:

            print("Registration error:", str(e))

            error\_message = "An unexpected error occurred during registration. Please try again later."

            if is\_ajax:

                return JsonResponse({

                    'success': False,

                    'message': {'general': error\_message}

                }, status=400)

            messages.error(request, error\_message)

    else:

        form = UserRegistrationForm()

    return render(request, 'authentication/register.html', {'form': form})

def logout\_view(request):

    logout(request)

    return redirect(reverse('authentication:login'))

# Dashboard Views

@login\_required

def dashboard(request):

    """

    Main dashboard view showing storage usage, recent activities, and notifications

    """

    # Calculate storage statistics

    total\_storage = 5 \* 1024 \* 1024 \* 1024  # 5GB in bytes

    used\_storage = File.objects.filter(user=request.user).aggregate(

        total=Sum('file\_size'))['total'] or 0

    storage\_percentage = (used\_storage / total\_storage) \* 100

    # Get recent activities

    recent\_activities = get\_recent\_activities(request.user)

    # Get notifications

    notifications = Notification.objects.filter(

        user=request.user,

        is\_read=False

    ).order\_by('-created\_at')[:5]

    # Get recent files

    recent\_files = File.objects.filter(

        user=request.user

    ).order\_by('-created\_at')[:5]

    context = {

        'storage\_used': {

            'used': used\_storage,

            'total': total\_storage,

            'percentage': storage\_percentage,

            'used\_formatted': format\_file\_size(used\_storage),

            'total\_formatted': format\_file\_size(total\_storage)

        },

        'collaborations': Collaboration.objects.filter(user=request.user).count(),

        'recent\_activities': recent\_activities,

        'recent\_files': recent\_files,

        'notifications': notifications,

        'unread\_notifications\_count': notifications.count(),

    }

    return render(request, 'authentication/dashboard/dashboard.html')

def get\_recent\_activities(user):

    """

    Helper function to get user's recent activities

    """

    # Get recent file activities

    recent\_files = File.objects.filter(user=user).order\_by('-modified\_at')[:5]

    activities = []

    for file in recent\_files:

        activities.append({

            'description': f"Updated file: {file.name}",

            'timestamp': file.modified\_at,

            'icon': 'file-earmark-text'

        })

    # Get recent collaborations

    recent\_collabs = Collaboration.objects.filter(user=user).order\_by('-created\_at')[:3]

    for collab in recent\_collabs:

        activities.append({

            'description': f"Started collaboration on: {collab.file.name}",

            'timestamp': collab.created\_at,

            'icon': 'people'

        })

    # Sort activities by timestamp

    activities.sort(key=lambda x: x['timestamp'], reverse=True)

    return activities[:5]  # Return only the 5 most recent activities

def format\_file\_size(size\_in\_bytes):

    """

    Helper function to format file sizes

    """

    for unit in ['B', 'KB', 'MB', 'GB']:

        if size\_in\_bytes < 1024.0:

            return f"{size\_in\_bytes:.1f} {unit}"

        size\_in\_bytes /= 1024.0

    return f"{size\_in\_bytes:.1f} TB"

@login\_required

def mark\_notification\_read(request, notification\_id):

    """

    AJAX view to mark a notification as read

    """

    try:

        notification = Notification.objects.get(id=notification\_id, user=request.user)

        notification.is\_read = True

        notification.save()

        return JsonResponse({'success': True})

    except Notification.DoesNotExist:

        return JsonResponse({'success': False}, status=404)

@login\_required

def mark\_all\_notifications\_read(request):

    """

    AJAX view to mark all notifications as read

    """

    Notification.objects.filter(user=request.user, is\_read=False).update(is\_read=True)

    return JsonResponse({'success': True})

**Urls.py**

# urls.py

from django.urls import path

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

from . import views

app\_name = 'authentication'

urlpatterns = [

    path('home/', views.home, name='home'),

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

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

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

    # Password Reset URLs

    path('password\_reset/',

         auth\_views.PasswordResetView.as\_view(

             template\_name='authentication/password\_reset\_form.html',

             success\_url='/auth/password\_reset/done/',  # Add this line

             email\_template\_name='authentication/password\_reset\_email.html',

             subject\_template\_name='authentication/password\_reset\_subject.txt'

         ),

         name='password\_reset'),

    path('password\_reset/done/',

         auth\_views.PasswordResetDoneView.as\_view(

             template\_name='authentication/password\_reset\_done.html'

         ),

         name='password\_reset\_done'),  # This name must match exactly

    path('reset/<uidb64>/<token>/',

         auth\_views.PasswordResetConfirmView.as\_view(

             template\_name='authentication/password\_reset\_confirm.html',

             success\_url='/auth/reset/done/'

         ),

         name='password\_reset\_confirm'),

    path('reset/done/',

         auth\_views.PasswordResetCompleteView.as\_view(

             template\_name='authentication/password\_reset\_complete.html'

         ),

         name='password\_reset\_complete'),

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

    path('notifications/mark-read/<int:notification\_id>/',

         views.mark\_notification\_read, name='mark\_notification\_read'),

    path('notifications/mark-all-read/',

         views.mark\_all\_notifications\_read, name='mark\_all\_notifications\_read'),

]