**Dashboard.html**

<!-- templates/dashboard/dashboard.html -->

{% extends 'authentication/dashboard/base.html' %}

{% load static %}

{% block content %}

<div class="container-fluid">

    <div class="row">

        <!-- Sidebar -->

        <div class="col-md-2 bg-light sidebar">

            <div class="sidebar-sticky pt-3">

                <div class="logo mb-4">

                    <h4><i class="fas fa-graduation-cap me-2"></i>

                        EduShare</h4>

                </div>

                <ul class="nav flex-column">

                    <li class="nav-item">

                        <a class="nav-link active" href="{% url 'authentication:dashboard' %}">

                            <i class="bi bi-house-door"></i> Dashboard

                        </a>

                    </li>

                    <li class="nav-item">

                        <a class="nav-link" href="#">

                            <i class="bi bi-file-earmark"></i> My Files

                        </a>

                    </li>

                    <li class="nav-item">

                        <a class="nav-link" href="#">

                            <i class="bi bi-people"></i> Collaborations

                        </a>

                    </li>

                    <li class="nav-item">

                        <a class="nav-link" href="#">

                            <i class="bi bi-share"></i> Shared Files

                        </a>

                    </li>

                    <li class="nav-item">

                        <a class="nav-link" href="#">

                            <i class="bi bi-gear"></i> Settings

                        </a>

                    </li>

                </ul>

            </div>

        </div>

        <!-- Main Content -->

        <main class="col-md-10 ms-sm-auto px-4">

            <div class="d-flex justify-content-between flex-wrap flex-md-nowrap align-items-center pt-3 pb-2 mb-3">

                <h1>Welcome back, Dr. {{ request.user.last\_name }}!</h1>

                <div class="search-bar">

                    <input type="text" class="form-control" placeholder="Search files, collaborations, or users...">

                </div>

            </div>

            <!-- Quick Stats Cards -->

<div class="row mb-4">

    <div class="col-md-3">

        <div class="card">

            <div class="card-body">

                <h5 class="card-title"><i class="fas fa-upload me-2"></i>Upload Files</h5>

                <p class="card-text">Share documents with your team</p>

                <button class="btn btn-primary">Quick Upload</button>

            </div>

        </div>

    </div>

    <div class="col-md-3">

        <div class="card">

            <div class="card-body">

                <h5 class="card-title"><i class="fas fa-users me-2"></i>Collaborations</h5>

                <p class="card-text">{{ collaborations }} active projects</p>

            </div>

        </div>

    </div>

    <div class="col-md-3">

        <div class="card">

            <div class="card-body">

                <h5 class="card-title"><i class="fas fa-database me-2"></i>Storage Used</h5>

                <div class="progress">

                    <div class="progress-bar" role="progressbar"

                         style="width: {{ storage\_used.percentage }}%"

                         aria-valuenow="{{ storage\_used.percentage }}"

                         aria-valuemin="0" aria-valuemax="100">

                    </div>

                </div>

                <p class="card-text mt-2">

                    {{ storage\_used.used|filesizeformat }} of {{ storage\_used.total|filesizeformat }}

                </p>

            </div>

        </div>

    </div>

    <div class="col-md-3">

        <div class="card">

            <div class="card-body">

                <h5 class="card-title"><i class="fas fa-clock me-2"></i>Recent Files</h5>

                <p class="card-text">View recent activities</p>

            </div>

        </div>

    </div>

</div>

            <!-- Recent Activities and Notifications -->

            <div class="row">

                <div class="col-md-8">

                    <div class="card">

                        <div class="card-body">

                            <div class="d-flex justify-content-between">

                                <h5 class="card-title">Recent Activities</h5>

                                <a href="#" class="text-primary">View All</a>

                            </div>

                            <div class="activities-list">

                                {% for activity in recent\_activities %}

                                <div class="activity-item">

                                    <div class="activity-icon">

                                        <i class="bi bi-{{ activity.icon }}"></i>

                                    </div>

                                    <div class="activity-details">

                                        <p>{{ activity.description }}</p>

                                        <small class="text-muted">{{ activity.timestamp|timesince }} ago</small>

                                    </div>

                                </div>

                                {% endfor %}

                            </div>

                        </div>

                    </div>

                </div>

                <div class="col-md-4">

                    <div class="card">

                        <div class="card-body">

                            <div class="d-flex justify-content-between">

                                <h5 class="card-title">Notifications</h5>

                                <a href="#" class="text-primary">Mark All Read</a>

                            </div>

                            {% for notification in notifications %}

                            <div class="notification-item">

                                <i class="bi bi-{{ notification.icon }}"></i>

                                <div class="notification-content">

                                    <p>{{ notification.message }}</p>

                                    <small class="text-muted">{{ notification.timestamp|timesince }} ago</small>

                                </div>

                            </div>

                            {% endfor %}

                        </div>

                    </div>

                </div>

            </div>

        </main>

    </div>

</div>

{% endblock %}

**Dashboard.js**

document.addEventListener('DOMContentLoaded', function() {

    // Initialize UI elements

    initializeSidebar();

    initializeUpload();

    initializeNotifications();

    updateStorageUsed();

    // Sidebar functionality remains the same

    function initializeSidebar() {

        const sidebarToggle = document.querySelector('.navbar-toggler');

        const sidebar = document.querySelector('.sidebar');

        if (sidebarToggle && sidebar) {

            sidebarToggle.addEventListener('click', function() {

                sidebar.classList.toggle('show');

                toggleSidebarOverlay();

            });

        }

    }

    // Overlay functionality remains the same

    function toggleSidebarOverlay() {

        let overlay = document.querySelector('.sidebar-overlay');

        if (!overlay) {

            overlay = document.createElement('div');

            overlay.className = 'sidebar-overlay';

            document.body.appendChild(overlay);

            overlay.addEventListener('click', function() {

                document.querySelector('.sidebar').classList.remove('show');

                overlay.classList.remove('show');

            });

        }

        overlay.classList.toggle('show');

    }

    // Improved file upload functionality

    function initializeUpload() {

        const quickUploadBtn = document.querySelector('.btn-primary');

        if (quickUploadBtn) {

            quickUploadBtn.addEventListener('click', function() {

                const fileInput = document.createElement('input');

                fileInput.type = 'file';

                fileInput.multiple = true;

                fileInput.accept = '\*/\*'; // You can restrict file types here

                fileInput.style.display = 'none';

                document.body.appendChild(fileInput);

                fileInput.click();

                fileInput.addEventListener('change', function(e) {

                    handleFileUpload(e.target.files);

                    document.body.removeChild(fileInput);

                });

            });

        }

    }

    function handleFileUpload(files) {

        if (!files.length) return;

        // Create progress indicator

        const progressDiv = document.createElement('div');

        progressDiv.className = 'upload-progress alert alert-info';

        progressDiv.innerHTML = `

            <div class="upload-status">Uploading ${files.length} file(s)...</div>

            <div class="progress">

                <div class="progress-bar" role="progressbar" style="width: 0%"></div>

            </div>

        `;

        document.querySelector('main').insertBefore(progressDiv, document.querySelector('main').firstChild);

        const formData = new FormData();

        Array.from(files).forEach(file => {

            formData.append('files', file); // Ensure this matches Django's view

        });

        // Get CSRF token from cookies

        const csrftoken = getCsrfToken();

        fetch('/auth/api/upload/', { // Correct URL to match your Django endpoint

            method: 'POST',

            headers: {

                'X-CSRFToken': csrftoken

            },

            body: formData

        })

        .then(response => {

            if (!response.ok) {

                throw new Error(`HTTP error! status: ${response.status}`);

            }

            return response.json();

        })

        .then(data => {

            if (data.status === 'success') {

                progressDiv.className = 'alert alert-success';

                progressDiv.innerHTML = `Successfully uploaded ${files.length} file(s)!`;

                // Refresh relevant UI sections

                updateRecentActivity();

                updateRecentFiles();

                updateStorageUsed();

            } else {

                throw new Error(data.message || 'An unknown error occurred.');

            }

        })

        .catch(error => {

            progressDiv.className = 'alert alert-danger';

            progressDiv.innerHTML = `Upload failed: ${error.message}`;

            console.error('Upload error:', error);

        })

        .finally(() => {

            // Remove the progress indicator after 3 seconds

            setTimeout(() => {

                progressDiv.remove();

            }, 3000);

        });

    }

    // Helper function to get CSRF token from cookies

    function getCsrfToken() {

        const name = 'csrftoken';

        let cookieValue = null;

        if (document.cookie && document.cookie !== '') {

            const cookies = document.cookie.split(';');

            for (let i = 0; i < cookies.length; i++) {

                const cookie = cookies[i].trim();

                if (cookie.substring(0, name.length + 1) === (name + '=')) {

                    cookieValue = decodeURIComponent(cookie.substring(name.length + 1));

                    break;

                }

            }

        }

        return cookieValue;

    }

    // Update recent activity with error handling

    function updateRecentActivity() {

        fetch('/auth/api/recent-activity/')

        .then(response => {

            if (!response.ok) throw new Error('Failed to fetch recent activity');

            return response.json();

        })

        .then(data => {

            const activityList = document.querySelector('.recent-activities');

            if (activityList && data.activities) {

                activityList.innerHTML = data.activities.map(activity => `

                    <div class="activity-item">

                        <div class="activity-icon">

                            <i class="bi bi-${activity.icon}"></i>

                        </div>

                        <div class="activity-details">

                            <p>${activity.description}</p>

                            <small class="text-muted">${activity.timestamp}</small>

                        </div>

                    </div>

                `).join('');

            }

        })

        .catch(error => console.error('Error updating activity:', error));

    }

    // New function to update recent files

    function updateRecentFiles() {

        fetch('/auth/api/recent-files/')

        .then(response => {

            if (!response.ok) throw new Error('Failed to fetch recent files');

            return response.json();

        })

        .then(data => {

            const filesList = document.querySelector('.recent-files');

            if (filesList && data.files) {

                filesList.innerHTML = data.files.map(file => `

                    <div class="file-item">

                        <div class="file-icon">

                            <i class="bi bi-file-earmark"></i>

                        </div>

                        <div class="file-details">

                            <p>${file.name}</p>

                            <small class="text-muted">${file.size} - ${file.uploaded\_at}</small>

                        </div>

                    </div>

                `).join('');

            }

        })

        .catch(error => console.error('Error updating recent files:', error));

    }

    // New function to update storage used

    function updateStorageUsed() {

        fetch('/auth/api/storage-usage/')

        .then(response => {

            if (!response.ok) throw new Error('Failed to fetch storage usage');

            return response.json();

        })

        .then(data => {

            const storageElement = document.querySelector('.storage-used');

            if (storageElement && data.usage) {

                storageElement.textContent = `${data.usage.used} of ${data.usage.total} used`;

            }

        })

        .catch(error => console.error('Error updating storage usage:', error));

    }

    // Notifications functionality remains the same

    function initializeNotifications() {

        const markAllReadBtn = document.querySelector('[data-action="mark-all-read"]');

        if (markAllReadBtn) {

            markAllReadBtn.addEventListener('click', function(e) {

                e.preventDefault();

                markAllNotificationsAsRead();

            });

        }

    }

});

// logout functionality

document.getElementById('logout-form').addEventListener('submit', function(e) {

    e.preventDefault();

    if (confirm('Are you sure you want to logout?')) {

        this.submit();

    }

});

// Add this to your dashboard's JavaScript file

// search functionalities

document.addEventListener('DOMContentLoaded', function() {

    const searchInput = document.querySelector('.search-input');

    const searchResults = document.querySelector('.search-results');

    let searchTimeout;

    searchInput.addEventListener('input', function(e) {

        clearTimeout(searchTimeout);

        const query = e.target.value;

        // Add loading state

        searchInput.classList.add('loading');

        // Debounce search requests

        searchTimeout = setTimeout(() => {

            if (query.length > 0) {

                fetch(`/auth/search-files/?query=${encodeURIComponent(query)}`)

                    .then(response => response.json())

                    .then(data => {

                        displaySearchResults(data.files);

                    })

                    .catch(error => console.error('Error:', error))

                    .finally(() => {

                        searchInput.classList.remove('loading');

                    });

            } else {

                searchResults.innerHTML = '';

                searchInput.classList.remove('loading');

            }

        }, 300);

    });

    function displaySearchResults(files) {

        if (files.length === 0) {

            searchResults.innerHTML = '<div class="p-3 text-center text-muted">No files found</div>';

            return;

        }

        const html = files.map(file => `

            <div class="search-result-item p-2 border-bottom">

                <div class="d-flex justify-content-between align-items-center">

                    <div>

                        <i class="bi bi-file-text me-2"></i>

                        <span>${file.name}</span>

                    </div>

                    <small class="text-muted">${file.size}</small>

                </div>

                <small class="text-muted">${file.created\_at}</small>

            </div>

        `).join('');

        searchResults.innerHTML = html;

    }

});

**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'),

    path('api/upload/', views.upload\_file, name='upload\_file'),

    path('api/recent-activity/', views.recent\_activity, name='recent\_activity'),

    path('api/recent-files/', views.recent\_files, name='recent\_files'),

    path('api/storage-usage/', views.storage\_usage, name='storage\_usage'),

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

    path('search-files/', views.search\_files, name='search\_files'),

]

**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.views.decorators.http import require\_http\_methods

from django.db.models import Sum

import json

from django.core.exceptions import ValidationError

from .forms import LoginForm, UserRegistrationForm

from .models import File, Collaboration, Notification

import mimetypes

def home(request):

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

# Login Views

@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})

# Register views

@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)

        # Add these debug prints

        print("Department in POST data:", request.POST.get('department'))

        if not form.is\_valid():

             # Add this debug print

            print("Department in cleaned\_data:", form.cleaned\_data.get('department'))

            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

            # Add this debug print

            print("Saved user department:", user.department)

            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})

# Logout views

@login\_required

@csrf\_protect

def logout\_view(request):

    if request.method == 'POST':

        logout(request)

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

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

# 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', context)

# Files upload views

@login\_required

@require\_http\_methods(["POST"])

def upload\_file(request):

    """

    Handle file uploads and create associated notifications

    """

    try:

        files = request.FILES.getlist('files')

        uploaded\_files = []

        for file in files:

            # Create File instance

            user\_file = File.objects.create(

                user=request.user,

                name=file.name,

                file=file,

                file\_size=file.size

            )

            # Create notification for the upload

            Notification.objects.create(

                user=request.user,

                message=f'Successfully uploaded file: {file.name}',

                icon='upload'

            )

            uploaded\_files.append({

                'id': user\_file.id,

                'name': user\_file.name,

                'size': format\_file\_size(user\_file.file\_size),

                'created\_at': user\_file.created\_at.isoformat()

            })

        return JsonResponse({

            'status': 'success',

            'files': uploaded\_files

        })

    except Exception as e:

        return JsonResponse({

            'status': 'error',

            'message': str(e)

        }, status=500)

@login\_required

def recent\_activity(request):

    """

    API endpoint to get recent activities

    """

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

    return JsonResponse({'activities': activities})

@login\_required

def recent\_files(request):

    """

    API endpoint to get recent files

    """

    files = File.objects.filter(user=request.user).order\_by('-created\_at')[:10]

    return JsonResponse({

        'files': [{

            'id': file.id,

            'name': file.name,

            'size': format\_file\_size(file.file\_size),

            'created\_at': file.created\_at.isoformat()

        } for file in files]

    })

@login\_required

def storage\_usage(request):

    """

    API endpoint to get storage usage 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

    return JsonResponse({

        'usage': {

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

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

            'percentage': storage\_percentage

        }

    })

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})

# profile views

@login\_required

def profile\_view(request):

    """View for displaying and updating user profile"""

    user = request.user

    if request.method == 'POST':

        if 'profile\_picture' in request.FILES:

            try:

                request.user.profile\_picture = request.FILES['profile\_picture']

                request.user.save()

                messages.success(request, 'Profile picture updated successfully!')

            except ValidationError as e:

                messages.error(request, str(e))

        return redirect('authentication:profile')

    context = {

        'user': request.user,

    }

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

# Search files views

@login\_required

def search\_files(request):

    """View for searching files"""

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

    if query:

        files = File.objects.filter(

            user=request.user,

            name\_\_icontains=query

        ).order\_by('-created\_at')

    else:

        files = File.objects.filter(user=request.user).order\_by('-created\_at')[:5]

    return JsonResponse({

        'files': [{

            'id': file.id,

            'name': file.name,

            'size': format\_file\_size(file.file\_size),

            'created\_at': file.created\_at.strftime('%Y-%m-%d %H:%M')

        } for file in files]

    })