1. A screenshot of a computer

   Description automatically generated



1. A screenshot of a computer

   Description automatically generated



1. A blue folder on a black background

   Description automatically generatedA screenshot of a computer

   Description automatically generated

testt

Views.py :

*def* upload\_folder(*request*):

if *request*.method == 'POST':

for key, file in *request*.FILES.items():

folder\_path, filename = os.path.split(key)

upload\_path = os.path.join(settings.MEDIA\_ROOT, folder\_path)

os.makedirs(upload\_path, *exist\_ok*=True)

with open(os.path.join(upload\_path, filename), 'wb+') as destination:

for chunk in file.chunks():

destination.write(chunk)

return JsonResponse({'message': 'Folders uploaded successfully!'})

return JsonResponse({'error': 'Invalid request'}, *status*=400)

# File & folder Upload

*class* FileUploadView(APIView):

permission\_classes = [IsAuthenticated]

*def* post(*self*, *request*):

try:

uploaded\_files = *request*.FILES.getlist('files')

folder\_paths = *request*.data.getlist('folders')

user\_id = *request*.user.id

# Handle folder creation

if folder\_paths:

for folder\_path in folder\_paths:

parent\_folder = None

folder\_hierarchy = folder\_path.split('/')

for folder in folder\_hierarchy:

folder\_instance, created = Folder.objects.get\_or\_create(

*name*=folder,

*parent\_folder*=parent\_folder,

*user\_id*=user\_id

)

parent\_folder = folder\_instance

# Return success if only folders are uploaded

if not uploaded\_files and folder\_paths:

return Response({"message": "Folders uploaded successfully!"}, *status*=201)

# Allowed extensions and max size

allowed\_extensions = ['jpg', 'jpeg', 'png', 'pdf', 'txt', 'docx', 'xlsx', 'csv', 'zip']

max\_size = 20 \* 1024 \* 1024 # 20 MB limit

uploaded\_file\_data = []

for uploaded\_file in uploaded\_files:

relative\_path = *request*.data.get(*f*"relative\_path\_{uploaded\_file.name}", uploaded\_file.name)

# Base upload directory

base\_upload\_dir = os.path.join(settings.MEDIA\_ROOT, 'uploads')

target\_folder = os.path.dirname(relative\_path)

final\_folder\_path = os.path.join(base\_upload\_dir, target\_folder)

os.makedirs(final\_folder\_path, *exist\_ok*=True)

# Validate file

file\_extension = uploaded\_file.name.split('.')[-1].lower()

if file\_extension not in allowed\_extensions:

return Response({"error": *f*"Invalid file type: {uploaded\_file.name}"}, *status*=400)

if uploaded\_file.size > max\_size:

return Response({"error": *f*"File size exceeds 20MB: {uploaded\_file.name}"}, *status*=400)

# Save File

safe\_file\_name = *f*"{slugify(os.path.basename(relative\_path))}.{file\_extension}"

file\_path = os.path.join(final\_folder\_path, safe\_file\_name)

with open(file\_path, 'wb+') as destination:

for chunk in uploaded\_file.chunks():

destination.write(chunk)

folder\_path = os.path.dirname(relative\_path)

parent\_folder = Folder.objects.filter(*user\_id*=user\_id, *name*=os.path.basename(folder\_path)).first()

file\_instance = File.objects.create(

*file*=*f*"uploads/{relative\_path}",

*file\_name*=uploaded\_file.name,

*size*=uploaded\_file.size,

*user\_id*=user\_id,

*file\_path*=file\_path,

*folder*=parent\_folder,

)

uploaded\_file\_data.append({

"file\_id": file\_instance.id,

"file\_name": file\_instance.file\_name

})

return Response({

"message": "Files and folders uploaded successfully!",

"files": uploaded\_file\_data

}, *status*=201)

except Exception as e:

print("Upload Error:", str(e))

return Response({"error": str(e)}, *status*=500)

# ViewSet for files

*class* FileViewSet(viewsets.ModelViewSet):

serializer\_class = FileSerializer

permission\_classes = [permissions.IsAuthenticated]

*def* get\_queryset(*self*):

return File.objects.filter(*user*=*self*.request.user, *is\_deleted*=False)

# List user-uploaded files

*class* FileListView(APIView):

permission\_classes = [IsAuthenticated]

*def* get(*self*, *request*):

files = File.objects.filter(*user\_id*=*request*.user.id)

serializer = FileSerializer(files, *many*=True)

return Response(serializer.data)

*class* FolderListView(APIView):

permission\_classes = [IsAuthenticated]

*def* get(*self*, *request*):

# Fetch folders and files

folders = Folder.objects.filter(*user\_id*=*request*.user.id)

files = File.objects.filter(*user\_id*=*request*.user.id, *is\_deleted*=False)

folder\_serializer = FolderSerializer(folders, *many*=True)

file\_serializer = FileSerializer(files, *many*=True)

# Combine both folders and files

return Response({

'folders': folder\_serializer.data,

'files': file\_serializer.data

})

#create folder

@method\_decorator(csrf\_exempt, *name*='dispatch')

*class* FolderView(APIView):

permission\_classes = [permissions.IsAuthenticated]

*def* post(*self*, *request*):

print("Request Data:", *request*.data) # Debugging

serializer = FolderSerializer(*data*=*request*.data)

if serializer.is\_valid():

serializer.save(*user*=*request*.user)

return Response(serializer.data, *status*=status.HTTP\_201\_CREATED)

print("Validation Errors:", serializer.errors) # Debugging

return Response(serializer.errors, *status*=status.HTTP\_400\_BAD\_REQUEST)

*class* FolderViewSet(viewsets.ModelViewSet):

queryset = Folder.objects.all()

serializer\_class = FolderSerializer

*def* create(*self*, *request*, \**args*, \*\**kwargs*):

serializer = *self*.get\_serializer(*data*=*request*.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data, *status*=status.HTTP\_201\_CREATED)

return Response(serializer.errors, *status*=status.HTTP\_400\_BAD\_REQUEST)

*class* FolderContentView(APIView):

permission\_classes = [permissions.IsAuthenticated]

*def* get(*self*, *request*, *folder\_id*):

try:

# Fetch subfolders and files in the folder

subfolders = Folder.objects.filter(*parent\_folder\_id*=*folder\_id*, *user*=*request*.user)

files = File.objects.filter(*folder\_id*=*folder\_id*, *user*=*request*.user, *is\_deleted*=False)

# Serialize results

folder\_serializer = FolderSerializer(subfolders, *many*=True)

file\_serializer = FileSerializer(files, *many*=True)

return Response({

'folders': folder\_serializer.data,

'files': file\_serializer.data

})

except Folder.DoesNotExist:

return Response({'error': 'Folder not found'}, *status*=status.HTTP\_404\_NOT\_FOUND)

*class* FolderListView(APIView):

permission\_classes = [IsAuthenticated]

*def* get(*self*, *request*, \**args*, \*\**kwargs*):

# Fetch folders and files

folders = Folder.objects.filter(*user*=*request*.user)

files = File.objects.filter(*user\_id*=*request*.user.id, *is\_deleted*=False)

folder\_serializer = FolderSerializer(folders, *many*=True)

file\_serializer = FileSerializer(files, *many*=True)

# Combine both folders and files in one response

return Response({

'folders': folder\_serializer.data,

'files': file\_serializer.data

})

*class* FileView(View):

*def* get(*self*, *request*, *file\_id*):

try:

# Retrieve the file instance from the database

file = File.objects.get(*id*=*file\_id*)

# Check if the 'file\_path' attribute is set and has a file

if not file.file\_path or not file.file\_path.name:

return HttpResponse("File not found", *status*=404)

file\_path = file.file\_path.path # This gets the absolute path of the

if os.path.exists(file\_path):

# Return the file as an attachment for download

response = FileResponse(open(file\_path, 'rb'), *as\_attachment*=True)

return response

else:

return HttpResponse("File not found", *status*=404)

except File.DoesNotExist:

return HttpResponse("File not found", *status*=404)

folder component ts (this is where button upload was and file & folder show here:

export *class* FolderComponent implements *OnInit* {

files: *any*[] = []; // Array to store file data

errorMessage: *string* = ''; // For displaying errors

folderFiles: File[] = [];

showStarredFiles: *boolean* = false; // To toggle between main and starred view

starredFiles: *any*[] = []; // Holds starred files

selectedFileId: *number* | *null* = null;

newFileName: *string* = '';

apiUrl: *string* = 'http://127.0.0.1:8000/api';

uploadDropdownVisible: *boolean* = false;

uploading = false; // Track if upload is in progress

progress: *number* | *null* = null; // Track progress

// Get references to file inputs

@ViewChild('fileInput') fileInput!: ElementRef;

@ViewChild('folderInput') folderInput!: ElementRef;

//properties file preview

showPreview: *boolean* = false;

loading: *boolean* = false;

fileType: *string* = '';

previewUrl: *string* = '';

fileSize: *string* = '';

// To track sort order for each column

sortOrder: { [*key*: *string*]: 'asc' | 'desc' } = {

name: 'asc',

size: 'asc',

modified: 'asc',

};

isAuthenticated: *boolean* = false;

folders: *any*;

currentFolderId: *string* | *undefined*;

*constructor*(

private *http*: HttpClient,

private *authService*: AuthService,

private *fileService*: FileService,

private *router*: Router,

private *uploadService*: UploadService,

) {}

ngOnInit(): *void* {

// Check if the user is authenticated by checking the token in AuthService

this.isAuthenticated = !!this.authService.getToken();

//this.fetchFiles(); // Loads all files

this.fetchFilesAndFolders();

this.fetchStarredFiles();

if (!this.isAuthenticated) {

this.errorMessage = 'You are not authenticated. Please log in.';

}

// Fetch all files only if authenticated

if (this.isAuthenticated) {

this.fetchFilesAndFolders();

} else {

this.errorMessage = 'You are not authenticated. Please log in.';

}

}

getHeaders() {

*const* token = this.authService.getToken();

return {

headers: new HttpHeaders({

'Authorization': `Token ${token}`,

'Content-Type': 'application/json', // Ensure Content-Type is application/json

}),

};

}

// Toggle dropdown visibility for a specific file

toggleDropdown(*file*: *any*): *void* {

file.showDropdown = !file.showDropdown;

}

// Utility to format the file size to a readable format

formatFileSize(*size*: *number*): *string* {

if (size < 1024) return `${size} B`;

if (size < 1048576) return `${(size / 1024).toFixed(2)} KB`;

if (size < 1073741824) return `${(size / 1048576).toFixed(2)} MB`;

return `${(size / 1073741824).toFixed(2)} GB`;

}

// Toggle upload dropdown

toggleUploadDropdown(): *void* {

this.uploadDropdownVisible = !this.uploadDropdownVisible;

}

// Trigger file input click for files

onFileUploadClick(): *void* {

this.fileInput.nativeElement.click();

}

// Trigger file input click for folders

onFolderUploadClick(): *void* {

this.folderInput.nativeElement.click();

}

// Handle File Upload (when selecting a file)

onUploadFile(*event*: *Event*): *void* {

*const* input = *event*.target as *HTMLInputElement*;

if (input.files) {

*const* files = *Array*.from(input.files); // Convert FileList to File[]

this.uploading = true;

this.uploadService.uploadFiles(files).subscribe(

(*event*) *=>* {

if (*event*.type === HttpEventType.UploadProgress) {

this.progress = Math.round((100 \* *event*.loaded) / (*event*.total || 1));

} else if (*event*.type === HttpEventType.Response) {

console.log('File uploaded successfully:', *event*.body);

this.uploading = false;

this.progress = null;

}

},

(*error*) *=>* {

console.error('Error uploading file:', *error*);

this.uploading = false;

this.progress = null;

});}}

// Handle Folder Upload (when selecting a folder)

onUploadFolder(*event*: *any*): *void* {

*const* files: *FileList* = *event*.target.files;

*const* folderData: { [*key*: *string*]: *any*[] } = {};

*Array*.from(files).forEach((*file*: File) *=>* {

*const* folderName = *file*.webkitRelativePath.split('/')[0];

if (!folderData[folderName]) {

folderData[folderName] = [];

}

folderData[folderName].push(*file*);

});

// Pass folderData to the upload service

this.uploadService.uploadFolders(folderData).subscribe(

(*response*) *=>* console.log('Upload success', *response*),

(*error*) *=>* console.error('Upload failed', *error*)

);

}

onCreateFolder(*parentFolderId*: *number* | *null* = null): *void* {

*const* folderName = prompt('Enter folder name:');

if (!folderName || folderName.trim() === '') {

alert('Folder name cannot be empty.');

return;

}

*const* payload = {

name: folderName.trim(),

parent\_folder: *parentFolderId* ?? null // Null if no parent

};

this.fileService.createFolder(payload).subscribe({

next: () *=>* {

alert('Folder created successfully!');

this.loadFolderContents(*parentFolderId*); // Reload contents

},

error: (*error*) *=>* {

console.error('Error creating folder:', *error*);

alert('Failed to create folder. Please try again.');

},

});

}

loadFolderContents(*folderId*: *number* | *null* = null): *void* {

*const* id = *folderId* ?? 0; // Default to 0 if null

this.fileService.getFolderContents(id).subscribe(

(*data*) *=>* {

// Process folders

*const* folders = data.folders.map((*folder*: *any*) *=>* ({

id: folder.id,

name: folder.name,

type: 'folder', // Explicitly mark as folder

parentFolder: folder.parent\_folder,

createdAt: folder.created\_at,

}));

// Process files

*const* files = data.files.map((*file*: *any*) *=>* ({

id: file.id,

name: file.file\_name,

size: file.size,

type: 'file', // Explicitly mark as file

createdAt: file.created\_at,

}));

// Combine folders and files for display

this.files = [...folders, ...files].map((*item*) *=>* ({

...item,

type: item.type || (item.id ? 'folder' : 'file') // Default to 'folder' if not explicitly a file

}));

},

(*error*) *=>* {

console.error('Error fetching folder contents:', *error*);

}

);

}

loadFiles(): *void* {

this.fileService.getFiles().subscribe(

(*files*) *=>* {

this.files = *files*;

},

(*error*) *=>* {

console.error('Error loading files:', *error*);

}

);

}

//open the file when click

onOpenFile(*file*: File, *event*: *Event*): *void* {

*event*.preventDefault(); // Prevent default behavior (if it's a link)

*const* token = this.authService.getToken(); // Retrieve the token from AuthService

if (!token) {

alert('You are not authenticated. Please log in.');

return;

}

// Generate the file URL based on file ID

*const* fileUrl = `http://127.0.0.1:8000/api/files/view/${file.id}/`;

// Open the file in a new tab

window.open(fileUrl, '\_blank');

}

//open folder

onOpenFolder(*folderId*: *string*): *void* {

// Find the folder and load its contents.

*const* folder = this.files.find(*file* *=>* file.id === folderId && file.isFolder);

if (folder) {

// You might need to set a `currentFolder` property and load its children.

this.currentFolderId = folderId;

this.files = folder.contents || []; // Assume `contents` contains the child files/folders.

}

}

async fetchFilesAndFolders() {

*const* token = this.authService.getToken();

try {

*const* response = await axios.get(`${this.apiUrl}/folders/`, {

headers: { Authorization: `Token ${token}` },

});

*const* data = response.data;

// Process folders

*const* folders = data.folders.map((*folder*: *any*) *=>* ({

id: *folder*.id,

name: *folder*.name,

type: 'folder', // Explicitly mark as folder

parentFolder: *folder*.parent\_folder,

modified: *folder*.created\_at,

}));

// Process files

*const* files = data.files.map((*file*: *any*) *=>* ({

id: *file*.id,

name: *file*.file\_name, // Adjust field to match backend

size: *file*.size,

type: 'file', // Explicitly mark as file

modified: *file*.created\_at,

}));

// Merge folders and files into a single list

this.files = [...folders, ...files];

} catch (error) {

console.error('Error fetching folders and files:', error);

}

}

//file review

formatBytes(*bytes*: *number*, *decimals* = 2): *string* {

if (*bytes* === 0) return '0 Bytes';

*const* k = 1024;

*const* dm = *decimals* < 0 ? 0 : *decimals*;

*const* sizes = ['Bytes', 'KB', 'MB', 'GB', 'TB'];

*const* i = Math.floor(Math.log(*bytes*) / Math.log(k));

return parseFloat((*bytes* / Math.pow(k, i)).toFixed(dm)) + ' ' + sizes[i];

}

// Function to preview file metadata

previewFile(*fileId*: *number*): *void* {

this.loading = true; // Start loading indicator

this.fileService.getFileMetadata(fileId).subscribe(

(*data*) *=>* {

this.previewUrl = data.url; // Set preview URL

this.fileType = data.type || 'application/octet-stream'; // Default MIME type

this.fileSize = this.formatBytes(*data*.size); // Format file size

this.showPreview = true; // Display the preview modal

this.loading = false; // Stop loading

},

(*error*: HttpErrorResponse) *=>* {

console.error('Error fetching file preview:', error.message);

this.loading = false; // Stop loading on error

}

);

}

}

Upload servise ts :

*class* UploadService {

private uploadUrl = 'http://localhost:8000/api/upload/';

*constructor*(private *http*: HttpClient) {}

uploadFiles(*files*: *File*[]): Observable<*any*> {

*const* formData = new *FormData*();

*files*.forEach(*file* *=>* {

formData.append('files', *file*, *file*.name); // Append each file to FormData

});

// POST request to upload files

return this.http.post(this.uploadUrl, formData, {

headers: new HttpHeaders(),

reportProgress: true,

observe: 'events',

});

}

uploadFolders(*folderData*: *any*): Observable<*any*> {

*const* formData = new *FormData*();

for (*const* folderName in *folderData*) {

*folderData*[folderName].forEach((*file*: *File*) *=>* {

formData.append(`${folderName}/${*file*.name}`, *file*);

});

}

return this.http.post(this.uploadUrl, formData);

}

}

url

router = DefaultRouter()

router.register(*r*'files', FileViewSet, *basename*='file')

router.register(*r*'folders', FolderViewSet, *basename*='folder')

urlpatterns = [

path('api/', include(router.urls)),

path('upload/', FileUploadView.as\_view(), *name*='file-upload'),

path('files/', FileListView.as\_view(), *name*='file-list'),

path('files/share/', views.share\_file, *name*='share\_file'),

path('files/view/<int:file\_id>/', FileView.as\_view(), *name*='file-view'),

path('upload/', FileUploadView.as\_view(), *name*='upload-file'),

path('file-preview/<int:file\_id>/', FilePreviewView.as\_view(), *name*='file\_preview'),

path('file-metadata/<int:file\_id>/', FileMetadataView.as\_view(), *name*='file\_metadata'),

path('folders/', views.FolderListView.as\_view(), *name*='folder-list'), # GET

path('folders/create/', views.FolderView.as\_view(), *name*='folder-create'), # POST

path('folders/<int:folder\_id>/', views.FolderContentView.as\_view(), *name*='folder\_list'),

path('folders/', FolderView.as\_view(), *name*='create-folder'),

path('upload-folder/', views.upload\_folder, *name*='upload\_folder'),

]