```
import { Component, OnInit } from "@angular/core";
import { CommonModule } from "@angular/common";
import { FormsModule } from "@angular/forms";
import { FileService } from "../../services/file.service";
interface SortConfig {
  key: "name" | "date" | "size";
  direction: "asc" | "desc";
}
@Component({
  selector: "app-drive",
  standalone: true,
  imports: [CommonModule, FormsModule],
  templateUrl: "./drive.component.html",
  styleUrls: ["./drive.component.scss"],
})
export class DriveComponent implements OnInit {
  files: any[] = [];
  diskDetails?: { free: number; size: number };
  currentPath: string[] = [];
  currentFolder: any[] = [];
  searchQuery: string = "";
  sortConfig: SortConfig = { key: "name", direction: "asc" };
  viewMode: "grid" | "list" = "grid";
  isUploading = false;
  isUploadMenuOpen = false;
  constructor(private fileService: FileService) {}
  ngOnInit() {
    this.loadFiles();
  }
  loadFiles() {
    this.fileService.getAllFiles().subscribe((response) => {
      this.files = response.allFilesAndDirs;
      this.diskDetails = response.diskDetails;
      this.updateCurrentFolder();
    });
  }
  updateCurrentFolder() {
    let current = this.files;
    for (const path of this.currentPath) {
      const folder = current.find((f) => f.name === path && f.isFolder);
      current = folder?.files || [];
    }
    this.currentFolder = this.sortFiles(this.filterFiles(current));
  }
  formatSize(bytes: number): string {
    const sizes = ["Bytes", "KB", "MB", "GB", "TB"];
  if (bytes === 0) return "0 Bytes";
  const i = Math.floor(Math.log(bytes) / Math.log(1024));
  return `${Math.round(bytes / Math.pow(1024, i))} ${sizes[i]}`;
}
filterFiles(items: any[]): any[] {
  if (!this.searchQuery) return items;
  return items.filter((item) =>
    item.name.toLowerCase().includes(this.searchQuery.toLowerCase()),
  );
}
sortFiles(items: any[]): any[] {
  return [...items].sort((a, b) => {
    if (this.sortConfig.key === "name") {
      return this.sortConfig.direction === "asc"
        ? a.name.localeCompare(b.name)
        : b.name.localeCompare(a.name);
    } else if (this.sortConfig.key === "date") {
      return this.sortConfig.direction === "asc"
        ? new Date(a.stats.mtime).getTime() -
            new Date(b.stats.mtime).getTime()
        : new Date(b.stats.mtime).getTime() -
            new Date(a.stats.mtime).getTime();
    } else if (this.sortConfig.key === "size") {
      return this.sortConfig.direction === "asc"
        ? a.stats.size - b.stats.size
        : b.stats.size - a.stats.size;
    }
    return 0;
  });
}
handleSort(key: "name" | "date" | "size") {
  if (this.sortConfig.key === key) {
    this.sortConfig.direction =
      this.sortConfig.direction === "asc" ? "desc" : "asc";
  } else {
    this.sortConfig = { key, direction: "asc" };
  }
  this.updateCurrentFolder();
}
async handleFileUpload(event: any) {
  const files = event.target.files;
  if (files.length === 0) return;
  this.isUploading = true;
  const formData = new FormData();
  Array.from(files).forEach((file) => {
    formData.append("files", file);
    });
    formData.append(
      "folderName",
      this.currentPath[this.currentPath.length - 1] || "root",
    );
    try {
      await this.fileService
        .uploadFiles(
          Array.from(files),
          this.currentPath[this.currentPath.length - 1] || "root",
        )
        .toPromise();
      this.loadFiles();
    } catch (error) {
      console.error("Upload failed:", error);
    } finally {
      this.isUploading = false;
      this.isUploadMenuOpen = false;
    }
  }
  navigateToFolder(folder: any) {
    this.currentPath.push(folder.name);
    this.updateCurrentFolder();
  }
  navigateUp() {
    this.currentPath.pop();
    this.updateCurrentFolder();
  }
  onSearchChange() {
    this.updateCurrentFolder();
  }
}
```