A PROJECT REPORT

on

"CLOUD MAKER"

submitted by

Mr. Vikas Kushwaha

Seat No :-

in partial fullfillment for the award of the degree

of

BACHELOR OF SCIENCE

in

COMPUTER SCIENCE

under the guidance of

Mrs. Swetha Iyer

Department of Computer Science

VIDYAVARDHINI'S
A. V. COLLEGE OF ARTS, K. M. COLLEGE OF COMMERCE
E. S. A. COLLEGE OF SCIENCE,
VASAI(WEST), PALGHAR-401208, MAHARASHTRA

(Sem V)

(2024-25)

ACKNOWLEDGEMENT

I would like to acknowledge my sincere thanks towards our project guide

Head of Computer Scince Department

Mrs. Srimathi Narayanan

for their valuable guidance and suggestions and

providing me an opportunity to do the project work in the college lab and

which made me complete the project successfully.

I am also thankful to

Mrs. Gyaneshwari Pawar

For providing such nice guidance in form of comments and corrections.

I am thakful to and fortunate enough to get contant encouragement,

support and guidance from all teaching staff of Computer Science

which helped us in successfully completing our project work.

Also, I would like to extend our sincere esteems to all staff in laboratory

for their timely support.

By **Vikas Kushwaha**,

T.Y.BSc (Computer Science)

DECLARATION

I **Vikas Kushwaha** hereby declare that,

The project entitled **"CLOUD MAKER"** submitted in the partial fulfillment for the award of Bachelor of Science in Computer Science during the academic year **2023 - 2024** is my original work and the project has not formed the basis for the award of any degree, associate ship, fellowhip or any other similar titles.

Signature of the Student:

Place:

Date:

PLAGARISM REPORT

GANTT CHART

TABLE OF CONTENT

INTRODUCTION

TITLE OF THE PROJECT: CLOUD MAKER

SYNOPSIS:

A cloud storage in distributed fashion.

This system is intended to be an alternative to online storage proveders like Google Drive. It's a distributed model where the user physically owns the resources in his/her own house. Unlike centralised cloud providers, the user is given a small computing device, preferrably an SBC like Rasberry Pi which acts as an 'Endpoint Device' and a gateway to access user's various media devices like Pen Drives, Hard Drives, Memory Cards, and any other sotrage media that can potentially interface with the Endpoint Device (Rasberry Pi, in our case.)

The user is intended to connect his Endpoint device using a Web Proxy which will be automatically setup acting as a Internet Gateway to his Endpoint Device. The Rasberry Pi will be primary product for the user that will act as a Cloud Storage Provider. He can access this Cloud Storage from any computer that has an Internet Connection. The user will be provided with a Web Interface from where he can view and manage all his Files and Folders.

In summary, it turns the user's own storage devices into a cloud making it convinient for the user to access his storage from anywhere in the world.

LIMITATIONS OF CURRENT SYSTEM

Thc cloud storage space have now become mainstream. People used store their files and data backups on External Storage Devices like Pend Drives and Hard Drives. However, these days we just upload all our content to cloud storage like Google Drive. This poses many problems and risks surrounding around Data Privacy, Security and Ownership of Data. It's now a well known fact that companies sell the User Data they harvest to other companies in exchange of profits.

There's also a problem with the costs involved. Cloud Storage Drives are extremely Expensive. Just for instance Google Cloud charges you Rs. 130 per month in India for a 100GB storage space for a single user. Assuming a year is just 10 months, it becomes Rs. 1300 for an year an Rs. 13,000 for a decade. That's actually quiet expensive for just a 100GB of storage space. A lot of people aren't really interested in spending a lot of money on such storage options as they often simply can't afford it. They will usually try to limit themselves by just using the limited storage space that is provided per account for free. 15 GigaBytes in case of Google Drive.

ADVANTAGES OF PROPOSED SYSTEM

This system tries to solve many of the problems with Cloud Storage proveders, as mentioned in the previous section that revolve araound data privacy and security and also costs.

Cloud Maker ensures that the data stays on user's physical medium ensuring that the user absolutely owns the data and no one else on the internet has access to it. Unless ofcourse, they had physical access to the device itself. This is incredibly useful for storing highly sensitive documents and other details that can be detremental for the user if fallen onto wrong hands. Being distributed in nature, it also minimizes the damage of data breaches. If a hacker do succeed in any case breaching the cloud storage, they only breache one device rather than breaching the whole community. This can be quite important for Government officials storing classified documents on the storage devices

Cloud Maker also makes it feasible for users to have large cloud storage space. As the user can simply use his/her own storage devices as a cloud store. The user can buy 1 TeraByte hard drive which will usually costs around Rs. 3,000 to 5,000 and can easily last for 6+ years and even a decade. This makes the costs and scale feasible for the user. This is especially useful for professionals like Video Editors and Graphics designers that often have Adboe project files spanning over multiple GigaBytes. They can't carry their hard drive everywhere and they often have to access various of their previous workd sporadically even for new projects.

TOOLS AND TECHNIQUES

This system involves an Endpoint Server and a client computer. For the endpoint server, let's assume a Rasberry Pi Model 3B flashed with a Linux Operating System like Debian 10 for ARM. This devices primarily runs two things, a web server and a set of scripts for managing storage devices. It then forwards it's Web Server Port to a VPS server using SSH. This VPS server has a pre-registered domain name and acts as a proxy to the Rasberry Pi Endpoint making it accessible throught the internet.

Initially, two shell scripts are started in the bacground – automountd and automount-clear. automountd monitors all the usb ports of Rasberry Pi and automatically mounts any storage device as soon as it's connected to it. It mounts them at a specific mount point (/media/user) which is later used by the Web Server. automount-clear monitors the mount point and cleans any dangling directories left over of unmount storage devices.

The Web Server makes the mount point accessible to other computers by providing a web interface for browsing and managing files. The Web Interface is essentially a File Manager. The Web Server is written in Go Programming Languages and uses http router from standard library which provides routing functionality and html/template (Go's built in template engine) which provides the building blocks of the web interface.

The Web Interface can be accessed through any computer or mobile device. The user can freely upload, download, or share files from his media device.

REQUIREMENT SPECIFICATION

1. **Hardware Requirements:**

   For Endpoint Server,

   - Rasberry PI Model B+ or newer
   - 16GB Memory Card
   - External Storage Drives of User Preferred Size

   For Client Device,

   - Connection to Endpoint Rasberry Pi Server (LAN / WAN)

2. **Software Requirements:**

   For Endpoint Server,

   - OS: Debian Raspi Linux
   - Shell: Bash
   - Programming Language: Go

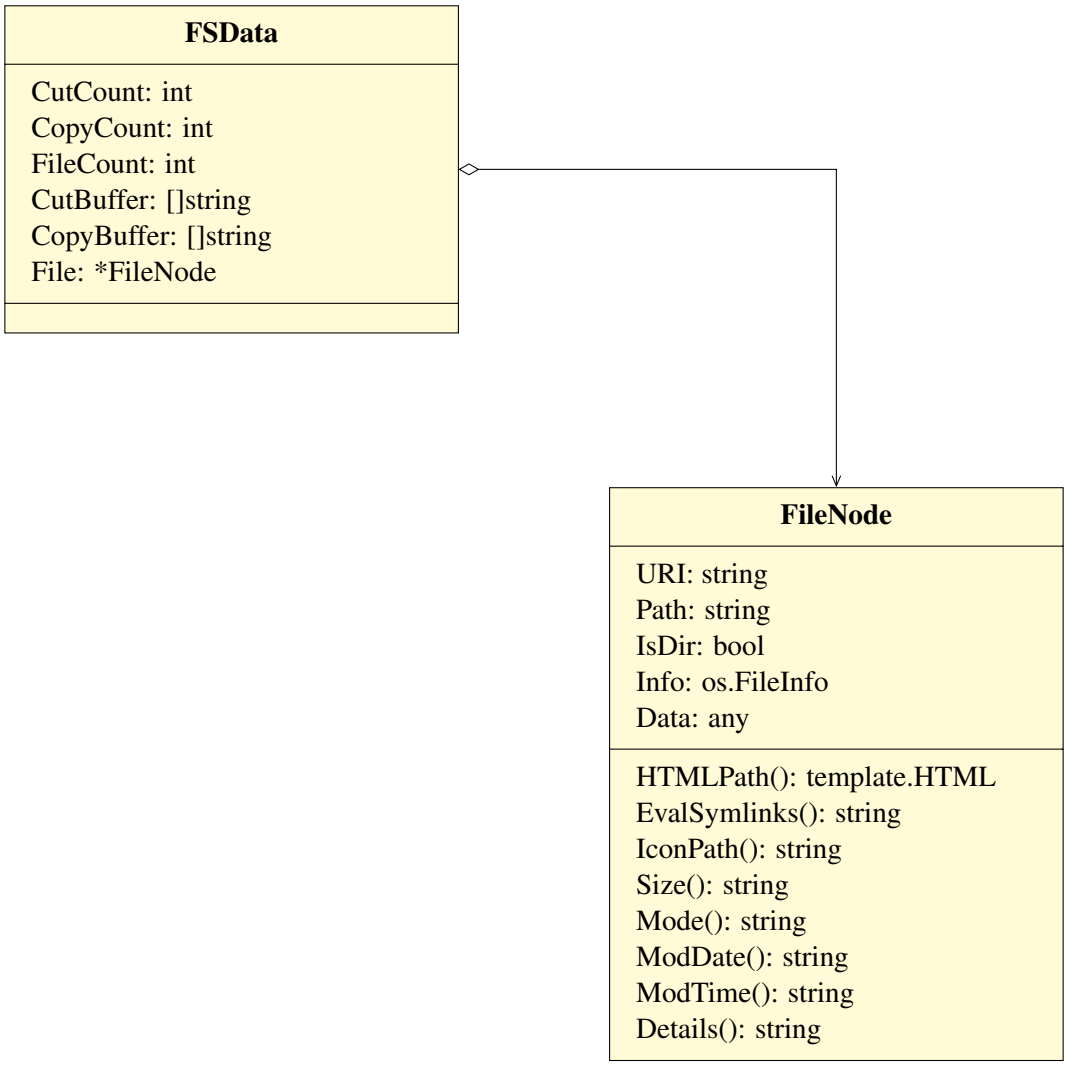
   For Client Device,

   - Any OS with latest Web Browser

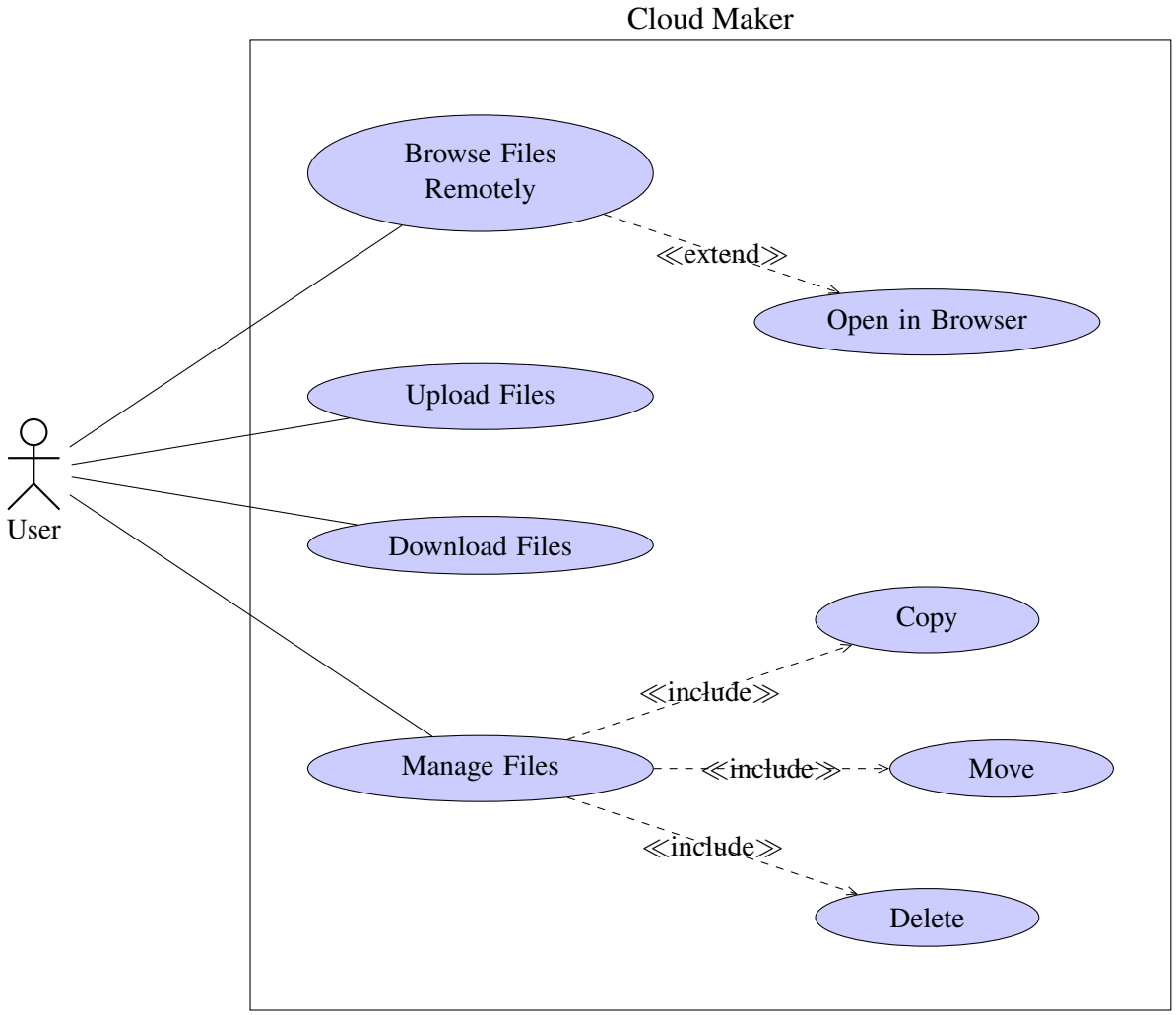
EVENT TABLE

Event	Trigger	Source	Activity	Response	Destination
Cut Files	User clicks on Cut Button	User	Files are added to Cut Buffer	Files in Cut Buffer	Endpoint Server
Copy Files	User clicks on Copy Button	User	Files are added to Copy Buffer	Files in Copy Buffer	Endpoint Server
Paste from Cut Buffer	User clicks on Paste button	User	Files are moved from Cut Buffer	Files Moved	Endpoint Server
Paste from Copy Buffer	User clicks on Paste button	User	Files are copied from Copy Buffer	Files Copied	Endpoint Server
Delete Files	User clicks on Delete Button	User	Selected Files are set to deletion	Confirm Deletion	Endpoint Server
Upload Local Files	User clicks on Upload Button	User	File Browser is opened for selection	User submits files	User
Download Remote Files	Server gets Download Requests	Endpoint Server	Server ZIPs requested file and sends to user	Compressed File recieved	User
Create Folder	User enters New Folder Name	User	New Folder is created on the system	Folder is shown	Endpoint Server

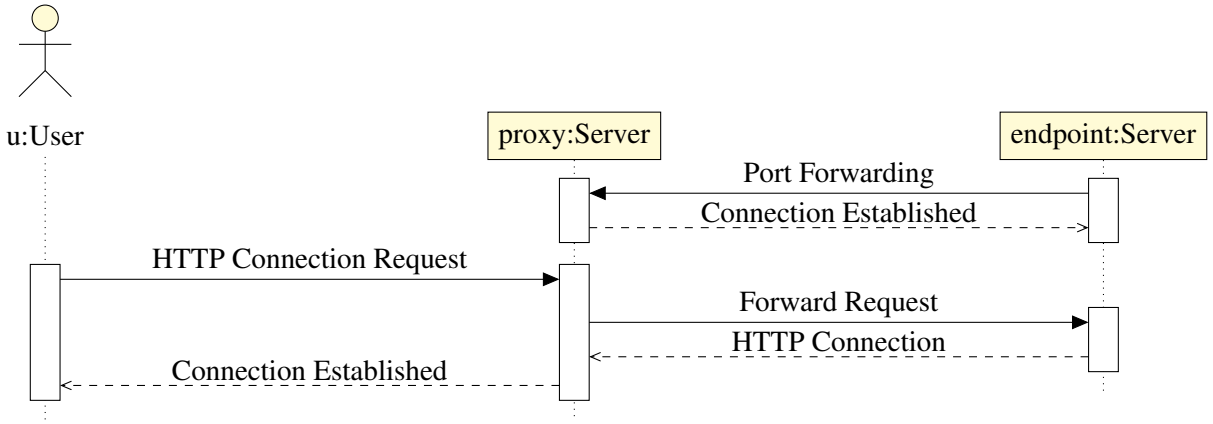
CLASS DIAGRAM

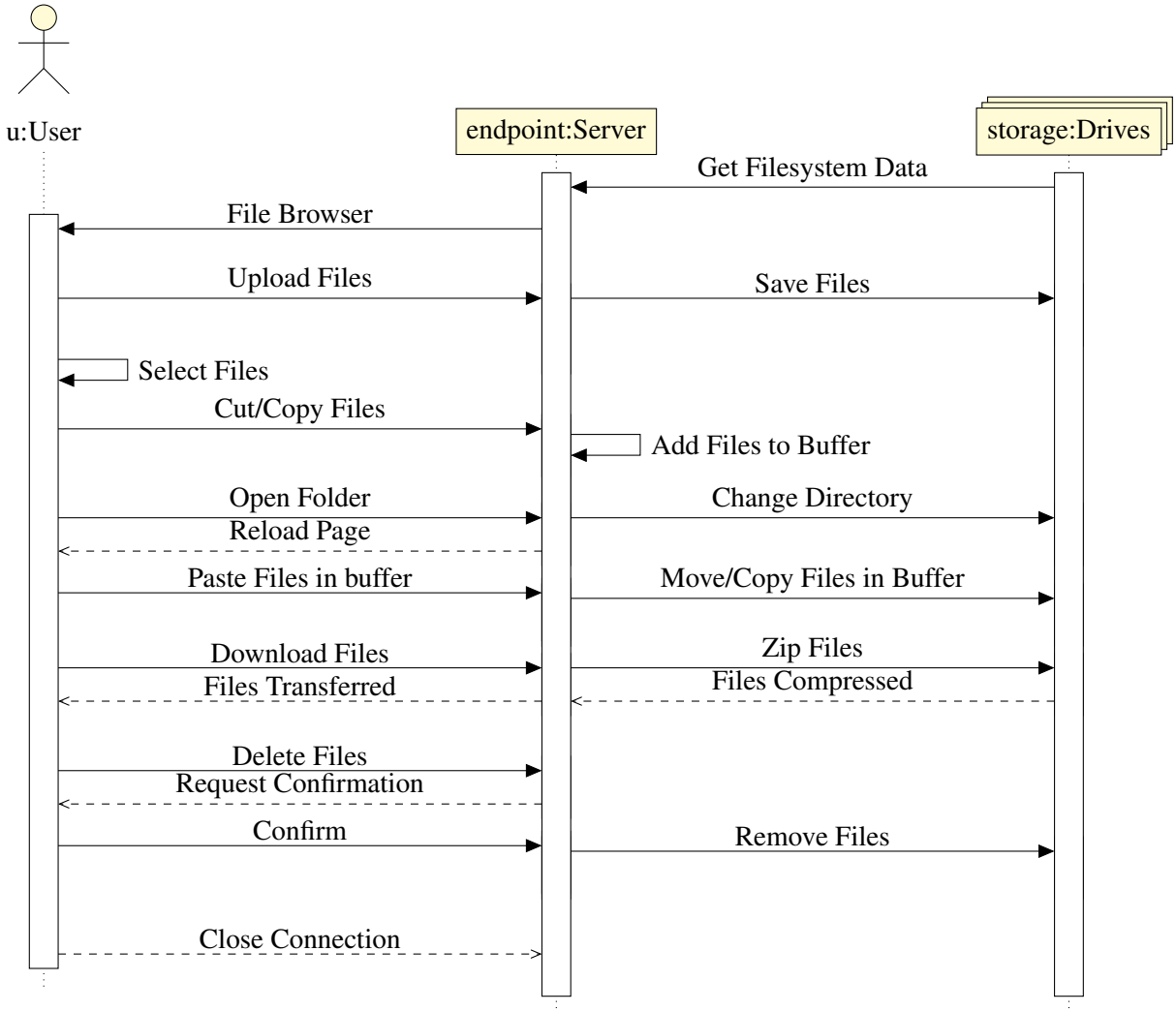


USE CASE DIAGRAM

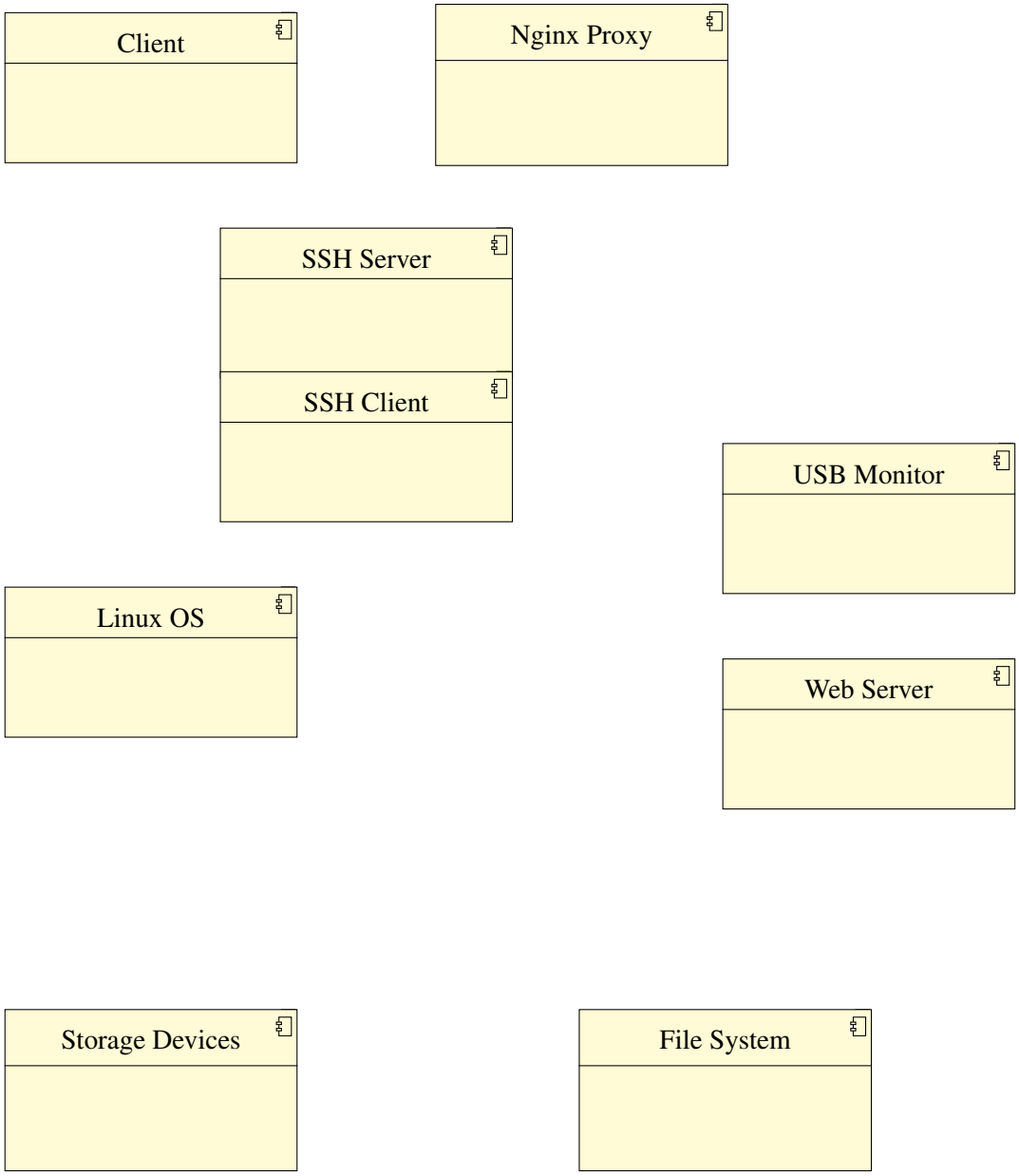


SEQUENCE DIAGRAM

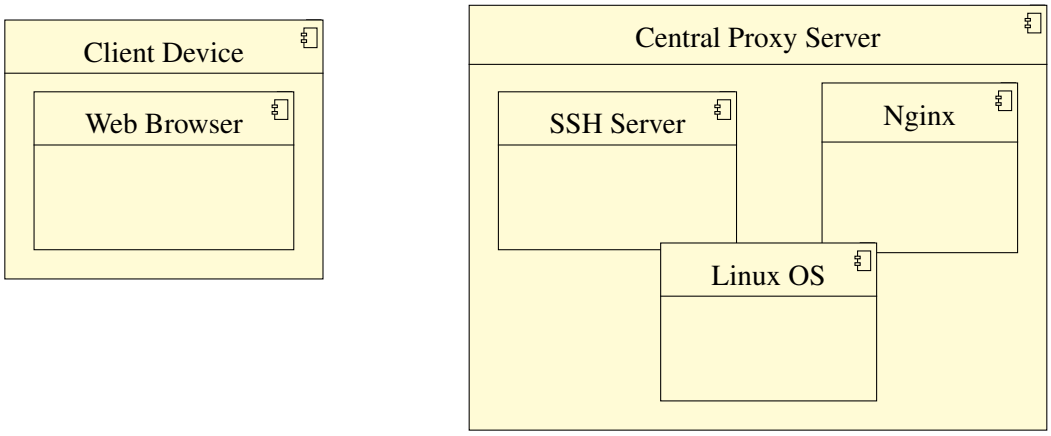


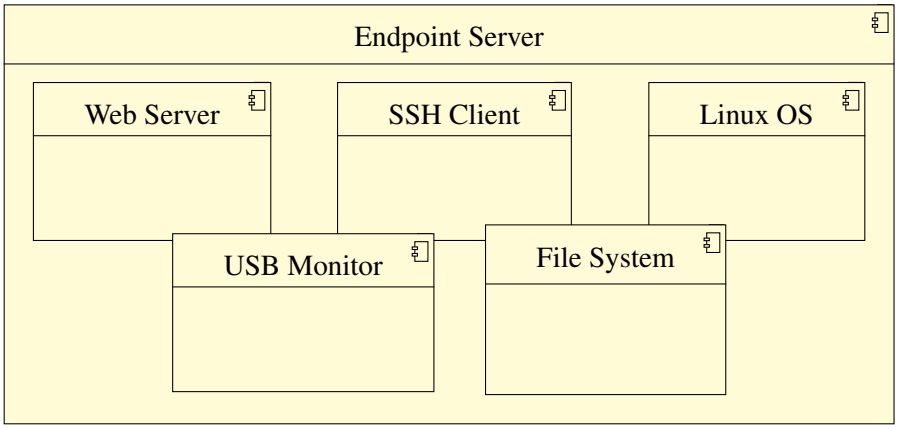


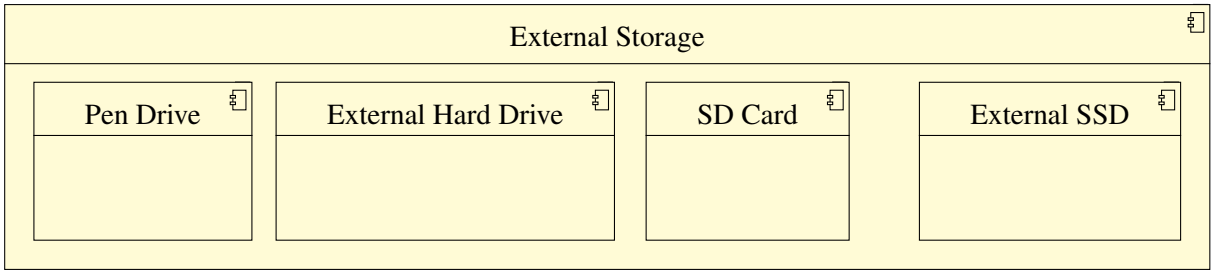
COMPONENT DIAGRAM



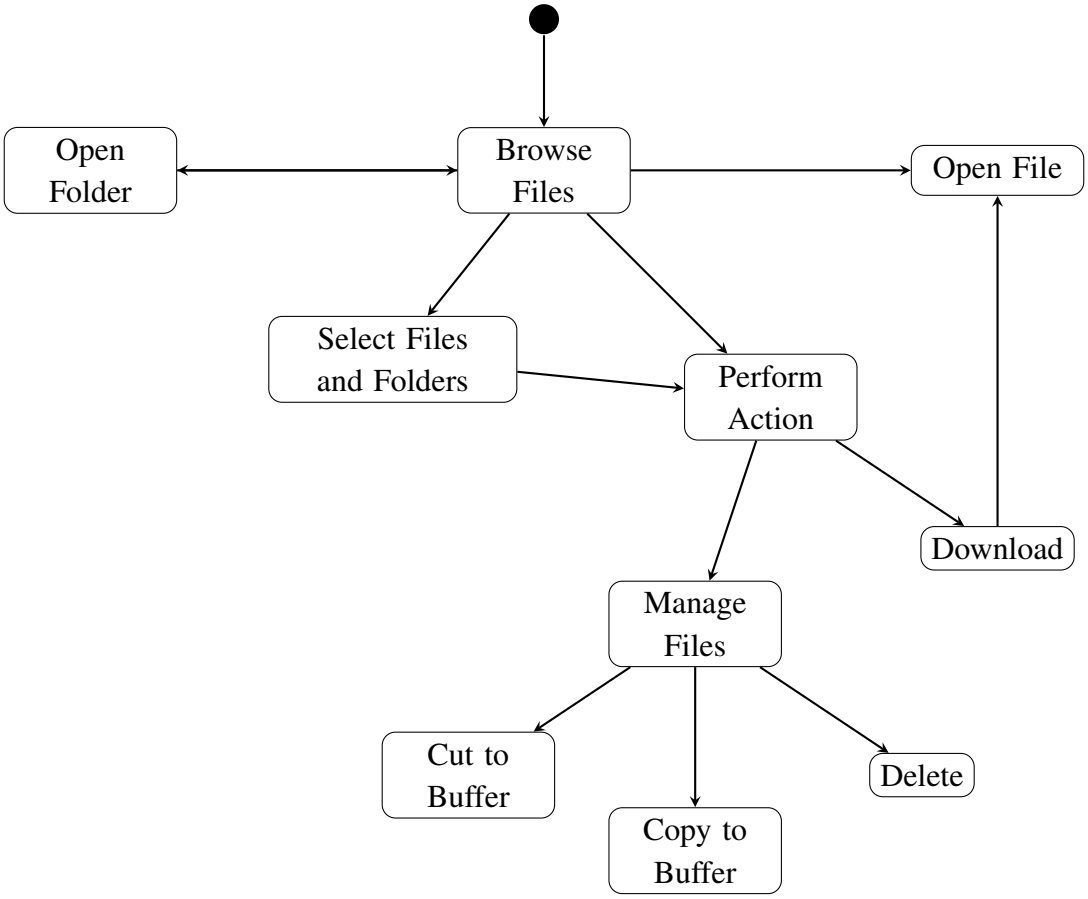
DEPLOYMENT DIAGRAM







ACTIVITY DIAGRAM



DATABASE

Representation of File Metadata in File System.

Attribute	DataType	Size	Retrieval
Name	Chars	255 Bytes	Primary Key
Permissions	Octal	4 Bytes	Not Null
User UID	Integer	4 Bytes	Not Null
Group GID	Integer	4 Bytes	Not Null
Modification Time	Time	4 Bytes	Not Null

SYSTEM IMPLEMENTATION

SERVER.GO

```
package main
    import (
    ”archive/zip”
    ”crypto/subtle”
    ”fmt”
    ”io”
    ”log”
    ”net/http”
    ”os”
    ”path/filepath”
    ”reflect”
    )
    func processAction(w http.ResponseWriter, r *http.Request, action string) *ServerError
    switch action
    default: return ServerErrornil, ”Invalid Action”, 400
    case ”cut”: return addSelectionToBuffer(w, r, cutBuffer)
    case ”copy”: return addSelectionToBuffer(w, r, copyBuffer)
    case ”cancel-cut”: return deleteBuffer(w, r, cutBuffer)
    case ”cancel-copy”: return deleteBuffer(w, r, copyBuffer)
    case ”cut-paste”: return moveFilesFromBuffer(w, r, cutBuffer)
    case ”copy-paste”: return pasteFilesFromBuffer(w, r, copyBuffer)
    case ”newdir”: return createNewDirectory(w, r)
    case ”delete”: return deleteSelectedFiles(w, r)
    func viewHandler(w http.ResponseWriter, r *http.Request) *ServerError
    var err error
    var serr *ServerError
    for k, v := range r.URL.Query()
    switch k
    default: http.Redirect(w, r, r.URL.Path, 302)
    case ”action”: return processAction(w, r, v[0])
    fileNode, serr := getFileNode(r.URL.Path)
    if serr != nil
    return serr
    if fileNode.Info.Mode()  os.ModeSymlink != 0
    fileURI := fileNode.URI
    target := ””
```

```
target, fileNode, err = fileNode.EvalSymlinks()
if err != nil
if !os.IsNotExist(err)
return ServerErrorerr, "", 500
if len(target) != 0
return ServerErrorerr, fileURI+": broken link to '"+target+"'", 404
else
return ServerErrorerr, fileURI+": Inaccessible link", 404
if !fileNode.IsDir
http.ServeFile(w, r, fileNode.Path)
return nil
dirList, err := getDirList(fileNode.Path, "name", true, true)
if err != nil
return ServerErrorerr, "", 404
cutBuf, err := readBuffer(cutBuffer)
if err != nil
return ServerErrorerr, "", 500
copyBuf, err := readBuffer(copyBuffer)
if err != nil
return ServerErrorerr, "", 500
fileNode.Data = dirList
err = renderTemplate(w, "viewDirList", FSData
CutCount: len(cutBuf),
CutBuffer: cutBuf,
CopyCount: len(copyBuf),
CopyBuffer: copyBuf,
FileCount: len(dirList),
File: fileNode,
)
if err != nil
return ServerErrorerr, "", 500
return nil
func downloadHandler(w http.ResponseWriter, r *http.Request) *ServerError
fmt.Printf("fileNode, files, serr := getSelectedNodes(r)
if serr != nil
return serr
if len(files) == 1  !files[0].IsDir
sendFile(w, r, files[0].Path)
return nil
zipName := fileNode.Info.Name() + ".zip"
```

target := ”/tmp/cloud/” + zipName
archive, err := os.Create(target)
if err != nil
return ServerErrorerr, ””, 500
defer archive.Close()
zipWriter := zip.NewWriter(archive)
defer zipWriter.Close()
for $, file := rangefiles$
err := addToZip(file.Path, zipWriter)
if err != nil
return ServerErrorerr, ””, 500
zipWriter.Close()
sendFile(w, r, target, zipName)
return nil
func uploadHandler(w http.ResponseWriter, r *http.Request) *ServerError
fileNode, serr := getFileNode(r.URL.Path)
if serr != nil
return serr
r.ParseMultipartForm(65536)
formData := r.MultipartForm
for $, handler := rangeformData.File[”attachments”]$
fmt.Printf(”fmt.Println(handler.Filename, ”:”, handler.Size)
file, err := handler.Open()
if err != nil
return ServerErrorerr, ””, 500
defer file.Close()
filepath := filepath.Join(fileNode.Path, handler.Filename)
fmt.Printf(”Saving to f, err := os.OpenFile(filepath, os.$O_WRONLY|os.O_CREATE, 0666$)
if err != nil
return ServerErrorerr, ””, 500
defer f.Close()
io.Copy(f, file)
fmt.Println(”Saved.”)
http.Redirect(w, r, ”/view/” + fileNode.URI, 303)
return nil
func fileHandler(w http.ResponseWriter, r *http.Request) *ServerError
fileNode, serr := getFileNode(r.URL.Path)
if serr != nil
return serr
if fileNode.IsDir

```
return ServerErrornil, ”File not Found.”, 404
http.ServeFile(w, r, fileNode.Path)
return nil
func handler(w http.ResponseWriter, r *http.Request) *ServerError
if r.URL.Path != ”/”
return ServerErrornil, ”Invalid URL”, 404
http.Redirect(w, r, ”view”, 303)
return nil
type httpHandler func(http.ResponseWriter, *http.Request) *ServerError
func (fn httpHandler) ServeHTTP(w http.ResponseWriter, r *http.Request)
username, password, ok := r.BasicAuth()
if !ok
w.Header().Add(”WWW-Authenticate”, ‘Basic realm=”restricted”, charset=”UTF-8”‘)
http.Error(w, ”Basic Auth Missing.”, 401)
return
realUsername, err := readData(”username”)
if err != nil
http.Error(w, ”Couldn’t retreive username data from server.”, 500)
realPassword, err := readData(”password”)
if err != nil
http.Error(w, ”Couldn’t retreive password data from server.”, 500)
usernameMatch := (subtle.ConstantTimeCompare([]byte(username[:]), realUsername[:])
== 1)
passwordMatch := (subtle.ConstantTimeCompare([]byte(password[:]), realPassword[:])
== 1)
if !usernameMatch —— !passwordMatch
w.Header().Set(”WWW-Authenticate”, ‘Basic realm=”restricted”, charset=”UTF-8”‘)
http.Error(w, ”Unauthrized”, http.StatusUnauthorized)
return
if serr := fn(w, r); serr != nil
if serr.Err != nil
fmt.Println(”Type:”, reflect.TypeOf(serr.Err))
fmt.Println(”Error Message:”, serr.Error())
if serr.Message == ””
serr.Message = ”Internal Server Error”
http.Error(w, serr.Message, serr.Status)
func main()
fileServer := http.FileServer(http.Dir(”./static”))
http.Handle(”/”, httpHandler(handler))
http.Handle(”/view/”, httpHandler(viewHandler))
```

```
    http.Handle("/upload/", httpHandler(uploadHandler))
    http.Handle("/download/", httpHandler(downloadHandler))
    http.Handle("/file/", httpHandler(fileHandler))
    http.Handle("/static/", http.StripPrefix("/static/", fileServer))
    fmt.Println("Listening on :8080")
    log.Fatal(http.ListenAndServe(":8080", nil))
```

HELPERS.GO

```
package main

import (
    "errors"
    "fmt"
    "net/http"
    "net/url"
    "os"
    "os/user"
    "path/filepath"
    "regexp"
    "strings"
)

type ServerError struct {
    Err error
    Message string
    Status int
}

func (e *ServerError) Error() string { return e.Err.Error() }
func (e *ServerError) Unwrap() error { return e.Err }


func getFileNode(URL string) (*FileNode, *ServerError) {
    path, err := url.PathUnescape(URL)
    if err != nil {
        return nil, &ServerError{err, "", 500}
    }
    p := strings.Split(path, "/")
    fileURI := strings.Trim(strings.Join(p[2:], "/"), "/")
```

```
    filePath := filepath.Join(homeDir, fileURI)

    fileInfo, err := os.Lstat(filePath)
    if err != nil {
        if errors.Is(err, os.ErrNotExist) {
            return nil, &ServerError{err, fileURI+" not found", 404}
        }
        return nil, &ServerError{err, "", 500}
    }

    return &FileNode{
        Path: filePath,
        URI: fileURI,
        IsDir: fileInfo.IsDir(),
        Info: fileInfo,
    }, nil
}


var filePattern = regexp.MustCompile(`^-file-entry--(.+)$`)

func getSelectedNodes(r *http.Request) (*FileNode, []*FileNode, *ServerError) {
    fileNode, e := getFileNode(r.URL.Path)
    if e != nil {
        return nil, nil, e
    }
    r.ParseMultipartForm(65536)
    fmt.Println(r.Form)
    var fileNames []string
    for key := range r.Form {
        if match := filePattern.FindStringSubmatch(key); len(match) > 1 {
            fileNames = append(fileNames, match[1])
        }
    }
    fmt.Printf("FileNames: %s\n", fileNames)
    if len(fileNames) == 0 {
        return fileNode, []*FileNode{fileNode}, nil
    }

    files := make([]*FileNode, len(fileNames))
```

```
    for i, fileName := range fileNames {
        fileNode, e := getFileNode( filepath .Join(r.URL.Path, fileName))
        if e != nil {
            return fileNode, nil, e
        }
        files [i] = fileNode
    }
    return fileNode, files , nil
}


func sendFile(w http.ResponseWriter, r *http.Request, info ... string ) {
    fmt. Printf ("info: %s\n", info [:])
    if len(info) < 2 {
        info = append(info, filepath .Base(info [0]) )
    }
    w.Header().Set("Content-Disposition", "attachment; filename=" + info [1])
    http. ServeFile(w, r, info [0])
}


func addSelectionToBuffer(w http.ResponseWriter, r *http.Request, bufferPath
     string ) *ServerError {
    _, files , e := getSelectedNodes(r)
    if e != nil {
        return e
    }
    buffer, err := readBuffer( bufferPath )
    if err != nil {
        return &ServerError{err, "", 500}
    }
    buff, err := os.OpenFile( bufferPath ,
        os.O_APPEND|os.O_WRONLY|os.O_CREATE, 0600)
    if err != nil {
        return &ServerError{err, "", 500}
    }
    var fileURI string

    writeToBuffer:
    for _, file := range files {
```

```
            fileURI = strings.Trim(file.URI, "/")
            for _, line := range buffer {
                if strings.Trim(line, "/") == fileURI {
                    continue writeToBuffer
                }
            }
            if file.IsDir {
                fileURI += "/"
            }
            buff.WriteString("/" + fileURI + "\r\n")
        }

        http.Redirect(w, r, r.URL.Path, 303)
        return nil
    }


    func deleteBuffer(w http.ResponseWriter, r *http.Request, bufferPath string)
        *ServerError {
        err := os.Remove(bufferPath)
        if err != nil {
            return &ServerError{err, "", 500}
        }
        http.Redirect(w, r, r.URL.Path, 303)
        return nil
    }


    func moveFilesFromBuffer(w http.ResponseWriter, r *http.Request, bufferPath
        string) *ServerError {
        fileNode, serr := getFileNode(r.URL.Path)
        if serr != nil {
            return serr
        }
        if !fileNode.IsDir {
            return &ServerError{nil, "Cannot move file, destination is not a
                directory", 400}
        }
        buffer, err := readBuffer(bufferPath)
        if err != nil {
```

```
            return &ServerError{err, "", 500}
        }
        for _, line := range buffer {
            err := copyTo(filepath.Join(homeDir, line), fileNode.Path)
            if err != nil {
                return &ServerError{err, "", 500}
            }
        }
        deleteBuffer(w, r, bufferPath)
        return nil
    }

    func pasteFilesFromBuffer(w http.ResponseWriter, r *http.Request, bufferPath
        string) *ServerError {
        fileNode, serr := getFileNode(r.URL.Path)
        if serr != nil {
            return serr
        }
        if !fileNode.IsDir {
            return &ServerError{nil, "Cannot copy files, destination is not a
                directory", 400}
        }
        buffer, err := readBuffer(bufferPath)
        if err != nil {
            return &ServerError{err, "", 500}
        }
        fmt.Printf("Buffer content: %s\n", buffer)
        for _, line := range buffer {
            err := copyTo(filepath.Join(homeDir, line), fileNode.Path)
            if err != nil {
                return &ServerError{err, "", 500}
            }
        }
        deleteBuffer(w, r, bufferPath)
        return nil
    }

    func createNewDirectory(w http.ResponseWriter, r *http.Request) *ServerError {
```

```
    fileNode, serr := getFileNode(r.URL.Path)
    if serr != nil {
        return serr
    }
    dirname := strings.TrimSpace(r.FormValue("newdir"))
    msg := "Requested to create directory '"+dirname+"' in "+fileNode.URI+"\n"
    if !fileNode.IsDir {
        msg = "Cannot create directory, the given destination is a file.\n" + msg
        return &ServerError{nil, msg, 400}
    }
    path := filepath.Join(fileNode.Path, dirname)
    isExist, err := fileExists(path)
    if isExist {
        msg = "Cannot create directory, a file with given name already exists.\n" +
            msg
        return &ServerError{nil, msg, 400}
    }
    err = os.Mkdir(path, 0755)
    if err != nil {
        return &ServerError{err, "", 500}
    }
    http.Redirect(w, r, r.URL.Path, 303)
    return nil
}


var (
    homeDir = "/media/"
    tempDir = "/tmp/cloud/"
    dataDir = "data"
    cutBuffer = filepath.Join(tempDir, "cut_buffer")
    copyBuffer = filepath.Join(tempDir, "copy_buffer")
)


func init() {
    err := os.MkdirAll(tempDir, 0755)
    if err != nil {
        panic(err)
    }
```

```
    home := os.Getenv("CLOUD_MAKER_HOME")
    if home != "" {
        isExist, err := fileExists(home)
        if err != nil {
            panic(err)
        }
        if isExist {
            homeDir = home
        }
        return
    }

    u, err := user.Current()
    if err != nil {
        panic(err)
    }
    homeDir += u.Username
}


func blockAction(w http.ResponseWriter, r *http.Request, action string)
    *ServerError {
    msg := ""
    _, files, serr := getSelectedNodes(r)
    if serr != nil {
        return serr
    }
    msg += "The " + action + " operation is currently disabled for testing and
        security reasons.\n"
    msg += "You requested to " + action + " following files :-\n\n"
    for _, file := range files {
        msg += file.Path + "\n"
    }
    fmt.Fprintf(w, msg)
    return nil
}


func deleteSelectedFiles(w http.ResponseWriter, r *http.Request) *ServerError {
    // return blockAction(w, r, "delete")
```

```
    fileNode, files, serr := getSelectedNodes(r)
    if serr != nil {
        return serr
    }
    for _, file := range files {
        if file.Path == homeDir {
            return &ServerError{nil, "Cannot delete root directory.", 400}
        }
        fmt.Printf("Deleting: %s\n", file.Path)
        err := os.RemoveAll(file.Path)
        if err != nil {
            return &ServerError{err, "", 500}
        }
        fmt.Println("Deleted.")
    }
    isExist, err := fileExists(fileNode.Path)
    if err != nil {
        return &ServerError{err, "", 500}
    }
    if !isExist {
        http.Redirect(w, r, "/view/" + filepath.Dir(fileNode.URI), 303)
    } else {
        http.Redirect(w, r, "/view/" + fileNode.URI, 303)
    }
    return nil
}
```

FILES.GO

```
package main
    import ( "archive/zip" "bufio" "io" "os" "fmt" "html/template" "net/http" "path/filepath"
"sort" "strconv" "strings" // "syscall" )
    type MalformedLinkError struct  Link string Target string
    func (e *MalformedLinkError) Error() string  return fmt.Sprintf("
    type FileNode struct  URI string Path string IsDir bool Info os.FileInfo Data any
    func (fileNode *FileNode) HTMLPath() template.HTML  var htmlpath string htmlpath
+= '¡a href="/view/' + '"¿' + "Home" + '¡/a¿ ' p := strings.Split(fileNode.URI, string(os.PathSeparator))
for i, dir := range p  if p[i] != ""  htmlpath += '¿ ¡a href="/view/' + filepath.Join(p[:i+1]...) +
'"¿' + dir + '¡/a¿ '  return template.HTML(htmlpath)
    func (fileNode *FileNode) EvalSymlinks() (string, *FileNode, error)  var err error target,
```

path, err := linkDeref(fileNode.Path); if err != nil if os.IsNotExist(err) return "", nil, err return target, nil, err fileInfo, err := os.Stat(path) if err != nil return target, nil, err return target, FileNode Path: path, URI: strings.TrimPrefix(path, homeDir), Info: fileInfo, IsDir: fileInfo.IsDir(), , nil

func (fileNode *FileNode) IconPath() (string, error) var icon string; switch fileNode.Info.Mode() os.ModeType default: icon = "file-earmark.svg" case os.ModeIrregular: icon = "question.svg" case os.ModeDir: icon = "folder2.svg" case os.ModeSymlink: $, fileNode, err := fileNode.EvalSymlinks()$ $nilif!os.IsNotExist(err)return"", erricon = "link - broken - 45deg.svg"elseiffileNode.IsDiricon =$

func (fileNode *FileNode) Size() (string, error) var err error if fileNode.Mode() == "l" $, fileNode, err = fileNode.EvalSymlinks()iferr! = nilreturn"", niliffileNode.IsDirreturn"", nilsiz$ $float64(fileNode.Info.Size())ifsize < 100returnstrconv.FormatFloat(size,' f', 0, 64) + "B", nilunit.$ $[]string"KB","MB","GB","TB","PB","EB","ZB"fori := 0; i < 7; i++size/ = 1024ifsize < 100r$ $"YiB", nil$

func (fileNode *FileNode) Mode() string switch fileNode.Info.Mode() os.ModeType default: return "f" case os.ModeDir: return "d" case os.ModeSymlink: return "l"

func (fileNode *FileNode) ModDate() string t := fileNode.Info.ModTime() return fmt.Sprintf("

func (fileNode *FileNode) ModTime() string t := fileNode.Info.ModTime() return fmt.Sprintf("

func (fileNode *FileNode) Details() (string, error) text := "Non-Regular File" switch fileNode.Info.Mode() os.ModeType

default: if fileNode.Info.Size() == 0 return "Empty File", nil file, err := os.Open(fileNode.Path) if err != nil return "", err buffer := make([]byte, 512) $, err = file.Read(buffer)iferr! =$ $nilreturn"", errcontentType := http.DetectContentType(buffer)ifcontentType ==$ $"application/octet - stream"text = "TextFile"elsetext = " * " + contentType$

case os.ModeDir: text = "Folder"

case os.ModeSymlink: target, $, err := fileNode.EvalSymlinks()iferr! = nilif!os.IsNotExist(err)re$

case os.ModeSocket: text = "Unix Socket"

case os.ModeDevice: text = "Device File"

case os.ModeNamedPipe: text = "Named Pipe"

case os.ModeTemporary: text = "Temporary File"

case os.ModeAppend: case os.ModeExclusive: case os.ModeSetuid: case os.ModeSetgid: case os.ModeCharDevice: case os.ModeSticky: case os.ModeIrregular:

return text, nil

func getDirSize(path string) (int64, error) var size int64 err := filepath.Walk(path, func($_s string, infoos.File$

func getDirList(path string, sortBy string, ascending bool, dirsFirst bool) ([]*FileNode, error) entries, err := os.ReadDir(path) if err != nil return nil, err files := make([]*FileNode, len(entries)) for i, entry := range entries filePath := filepath.Join(path, entry.Name()) fileURI := strings.TrimLeft(path, homeDir) fileInfo, err := entry.Info() if err != nil return nil, err files[i] = FileNode Path: filePath, URI: fileURI, IsDir: entry.IsDir(), Info: fileInfo,

switch sortBy case "name": sort.SliceStable(files, func(i, j int) bool return strings.ToLower(files[i].Info.Na ¡ strings.ToLower(files[j].Info.Name())) case "size": sort.SliceStable(files, func(i, j int) bool

return files[i].Info.Size() ¡ files[j].Info.Size()) case ”time”: sort.SliceStable(files, func(i, j int) bool return files[i].Info.ModTime().Before(files[j].Info.ModTime()))

if !ascending for i, j := 0, len(files)-1; i ¡ j; i, j = i+1, j-1 files[i], files[j] = files[j], files[i]

if dirsFirst var dirs, notDirs []*FileNode for , $fileNode := rangefilesinfo, err := os.Stat(fileNode.Pa$

return files, nil

func addToZip(source string, writer *zip.Writer) error return filepath.Walk(source, func(path string, info os.FileInfo, err error) error if err != nil return err header, err := zip.FileInfoHeader(info) if err != nil return err header.Method = zip.Deflate header.Name, err = filepath.Rel(filepath.Dir(source), path) if err != nil return err if info.IsDir() header.Name += ”/” headerWriter, err := writer.CreateHeader(header) if err != nil return err if !info.Mode().IsRegular() return nil f, err := os.Open(path) if err != nil return err defer f.Close()

, $err = io.Copy(headerWriter, f)returnerr)$

func readBuffer(path string) ([]string, error) buff, err := os.OpenFile(path, os.O$_R DONLY|os.O_C REATE,$ $nilreturnnil, errdeferbuff.Close()$

var buffer []string scanner := bufio.NewScanner(buff) for scanner.Scan() buffer = append(buffer, scanner.Text()) return buffer, nil

func fileExists(path string) (bool, error) , $err := os.Lstat(path)iferr == nilreturntrue, nilifos.IsNotE$

func copyFile(src, dst string) error fin, err := os.Open(src) if err != nil return err defer fin.Close()

fout, err := os.Create(dst) if err != nil return err defer fout.Close()

, $err = io.Copy(fout, fin)iferr! = nilreturnerrfin.Close()$

return nil

func copyTo(src, dstDir string) error info, err := os.Lstat(src) if err != nil return err dst := filepath.Join(dstDir, info.Name())

fmt.Printf(”Copying switch info.Mode() os.ModeType case os.ModeDir: if err := os.MkdirAll(dst, 0755); err != nil return err if err := copyDir(src, dst); err != nil return err case os.ModeSymlink: if err := copySymlink(src, dst); err != nil return err default: if err := copyFile(src, dst); err != nil return err fmt.Println(”Finished Copying.”)

/* stat, ok := info.Sys().(*syscall.Stat$_t$)$if!okreturnfmt.Errorf(”failedtogetrawsyscall.Stat_tdatafor'i$ $os.Lchown(dst, int(stat.Uid), int(stat.Gid)); err! = nilreturnerr * /$

if info.Mode()os.ModeSymlink == 0 return os.Chmod(dst, info.Mode()) return nil

func linkDeref(link string) (string, string, error) target, err := os.Readlink(link) if err != nil return ””, ””, err path := target if filepath.IsAbs(target) if !strings.HasPrefix(path, homeDir) return target, ””, os.ErrNotExist target = strings.TrimPrefix(target, homeDir) else path = filepath.Join(filepath.Dir(link), path) if !strings.HasPrefix(path, homeDir) return target, ””, os.ErrNotExist return target, path, nil

func readData(name string) ([]byte, error) data, err := os.ReadFile(filepath.Join(dataDir, name)) if err != nil return nil, err return data[:len(data)-1], nil

func writeData(name string, data string) error return os.WriteFile(filepath.Join(dataDir, name), []byte(data), 644)

func init() err := os.MkdirAll(dataDir, 0755) if err != nil panic(err)

TEMPLATES.GO

package main

import (”net/http” ”html/template”)

type FSData struct CutCount int CopyCount int FileCount int CutBuffer []string CopyBuffer []string File *FileNode

var templates = make(map[string]*template.Template)

func renderTemplate(w http.ResponseWriter, tmpl string, data any) error return templates[tmpl].ExecuteTemplate(w, ”base.html”, data)

func init() templates[”viewDirList”] = template.Must(template.New(”viewDirList.html”,).ParseFiles(”templates/base.html”, ”templates/viewDirList.html”))

AUTOMOUNTD

!/bin/sh

pathtoname() udevadm info -p /sys/”1”$|awk - vFS ==' /DEVNAME/print2$'

stdbuf -oL – udevadm monitor –udev -s block — while read -r – $_{eventdevpath;doif[}$”event” = add]; then devname=$(pathtoname$”devpath”) udisksctl mount –block-device ”$devname$” $--no - user - interaction$

target=”$(lsblk - noLABEL$”devname”)” [-z ”$target$”]$target = $”(lsblk -no UUID ”$devname$”)”$[-z$”target”] continue sudo mount -v –mkdir ”$devname$”” $/media/root/$target” mkdir -pv ”/media/master/$target$” $sudobindfs - u$(id -u) -g $(id - g)$”$/media/root/$target” ”/media/master/$target$” $fidone$

AUTOMOUNT-CLEAR

!/bin/sh

find /mnt -mindepth 1 -maxdepth 1 -printf ”lsblk — grep -F ”$path$” $done$

RESULTS

CONCLUSION

REFERENCES