Skybin

CSCI U511 Team Project
Final Report

Joshua Greeff, Josh Frazier, Michael Sinclair

December 8, 2023

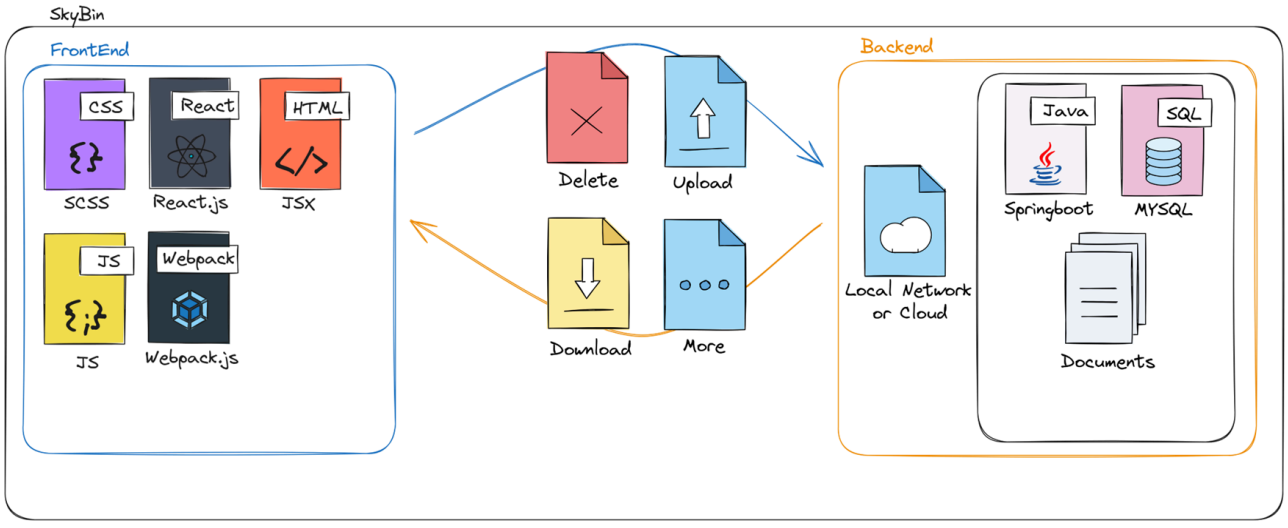
Table Of Contents

Final Project Description

Proposal: Network attached storage served over HTTP with user authentication and authorization. Users will be able to upload, download, and manage files and folders in their own user directory. A user-friendly web interface will be developed using Javascript with the backend written using Java. User data (username, password, file ID) will be stored in a local MySQL database. The files themselves will be saved on the host device. Class concepts that will be covered in the project are multithreading, child process management, exception handling, etc.

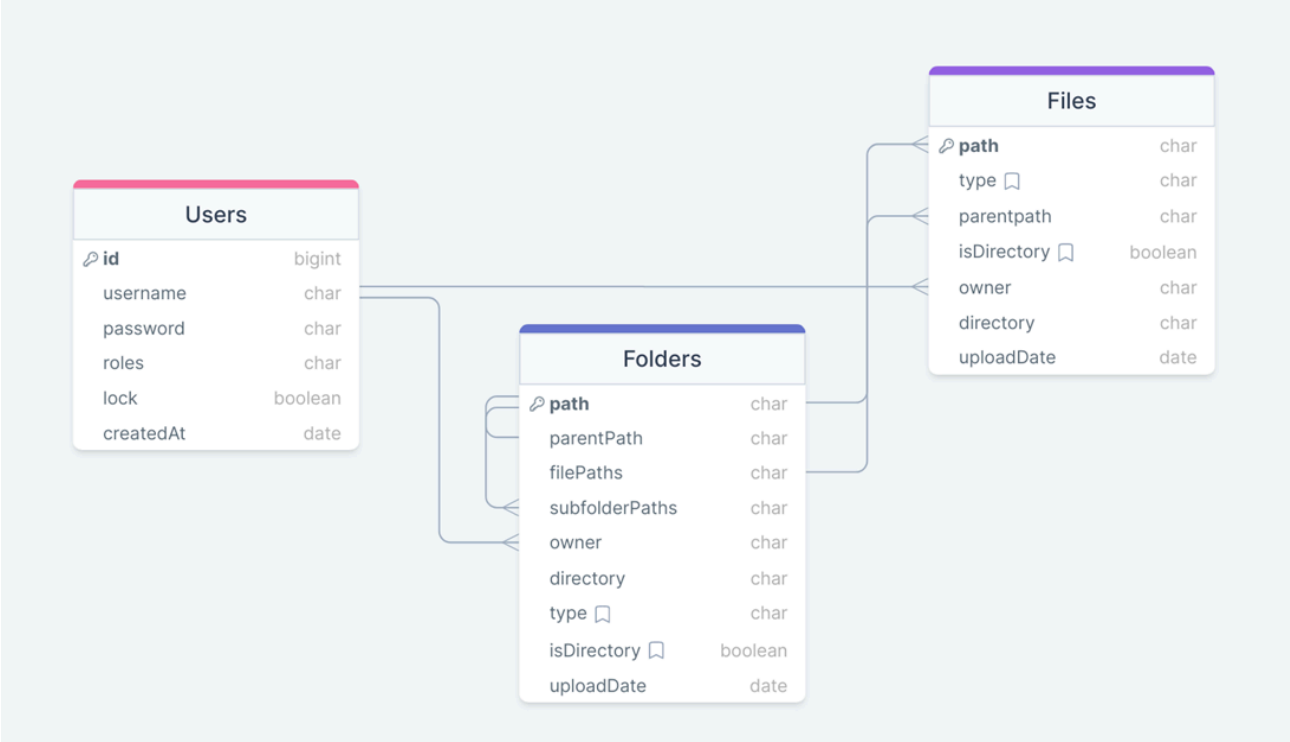
Project Details

Software Architecture Overview



For our application, we created a Spring Boot Maven project of which the backend would be written in Java. User account data and file information is hosted in a locally-run MySQL server that is accessed by the backend using JPA and a JDBC driver. The frontend was created using React as a framework and written using Javascript, CSS, HTML and utilized libraries such as Webpack and Axios. The frontend that is served to the client interacts with the backend using a REST API that receives HTTP requests with a JSON Web Token and a JSON formatted body. The frontend also receives information via the response to these HTTP requests.

MySQL Server



Pictured above is an entity-relationship diagram that covers the structure of our MySQL server. The database stores all information related to users, folders, and files. There are several relationships between these, most notably of which is the many-to-one relationship between users/folders and users/files as well as folders/files. Since SQL statements are not always atomic, we used thread locks to handle concurrent requests and avoid race conditions. As previously mentioned, the local MySQL server is accessed using JPA with a JDBC driver and only accessible by the backend's repository classes. When information is pulled from the database, it is mapped to a Java object as modeled by our model classes which allows us to interact with, manipulate, and process data as needed. One more thing to note, files are not hosted on the MySQL server, they are hosted in their own directory on the device hosting the server. The MySQL server instead documents file information and indexes so that the files may be properly retrieved and stored.

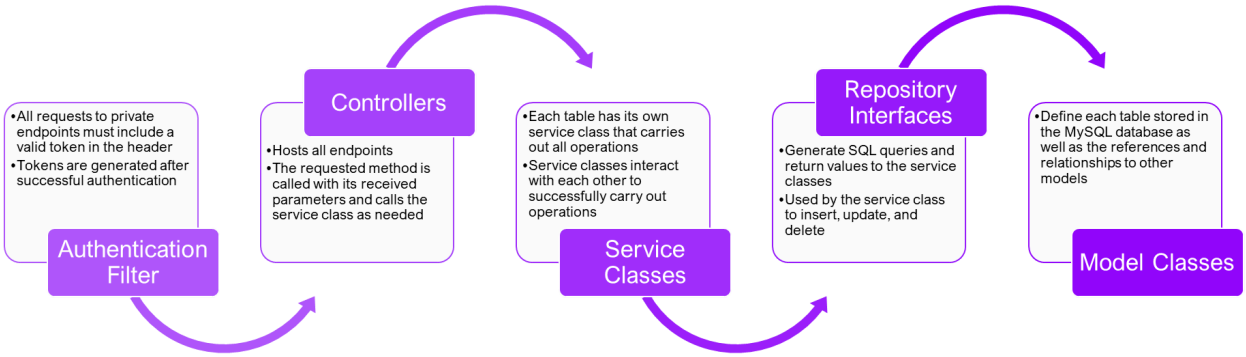
Backend

As previously mentioned, the backend is interacted with through a REST API and HTTP requests/responses. Below is a table briefly covering some of our endpoints as well as a brief description of each HTTP method we used.

/api/refresh /api/user/download	GET	Request data from the server

/api/user/listFiles		
/api/create /api/authenticate /api/user/upload	POST	Send data to the server
/api/delete /api/user/deleteFile /api/user/deleteFolder	DELETE	Delete data on the server
/api/update/username /api/user/renameFile /api/user/renameFolder	PUT	Update data on the server

All endpoints past the login page are protected and require a valid JWT to access



Pictured above is the general structure of the backend, as well as the path that HTTP requests take as they are received.
For our security, we used Spring Security and only allow the frontend to be served publicly, and only the /api/create and /api/authenticate POST endpoints are public so that users may create an account and login to receive a valid token. All other endpoints require authentication using the token returned by /api/create and /api/authenticate. These tokens are signed using a secret key and HMACSHA256, and their body contains the user's username, claims, and the token's expiration date, all formatted in JSON.

Concurrency
A server that only allows one request at time is not very useful, so our application can support up to 200 simultaneous requests, as defined here:

```
@ConfigurationProperties(prefix = "server", ignoreUnknownFields = true)
public class ServerProperties {
public static class Tomcat {
public static class Threads {
private int max = 200; // Maximum amount of worker threads }}}
```

Each web request creates a new thread that carries out the request. Spring Boot uses asynchronous programming and thread pools to handle these simultaneously. To avoid race conditions and request conflicts, each user account has a lock saved in the database that must be obtained by each thread to make any account of file/folder changes.

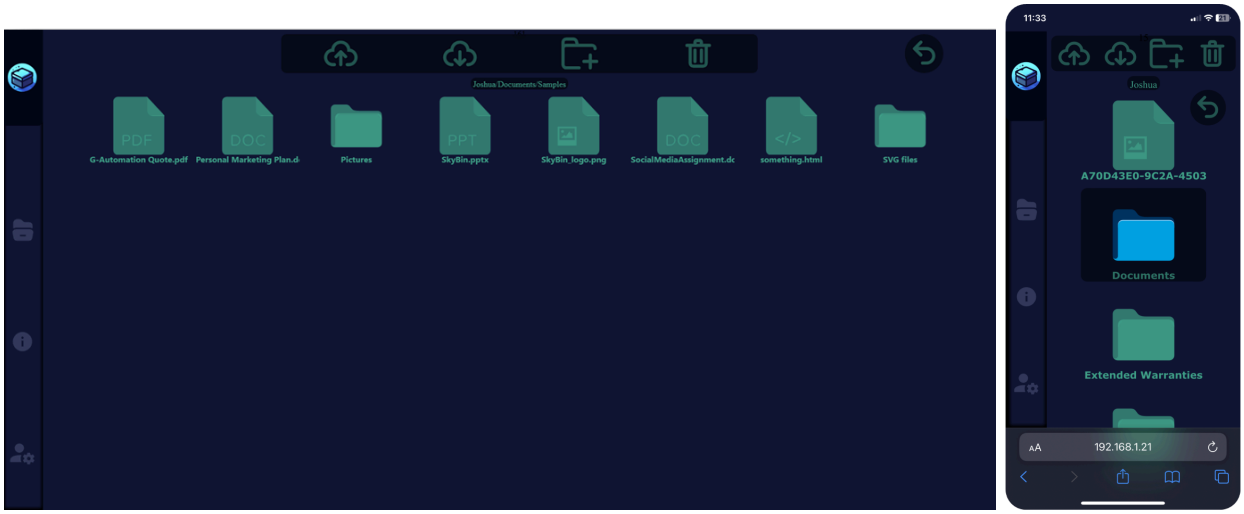
```
@Override
public Boolean lock(String username) {
    Optional<UserInfo> user = userRepository.findByUsername(username);
    if (user.isPresent()) {
        while (user.get().getLock()) {
            try {
                Thread.sleep(1000);
                user = userRepository.findByUsername(username);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        user.get().setLock(true);
        userRepository.save(user.get());
        return true;
    }
    return false;
}

@Override
public Boolean unlock(String username) {
    Optional<UserInfo> user = userRepository.findByUsername(username);
    if (user.isPresent()) {
        user.get().setLock(false);
        userRepository.save(user.get());
        return true;
    }
    return false;
}
```

Above is our implementation of locks, and every request must first obtain the lock for the user's account before making any changes.

Frontend

We used JavaScript with React.js, Webpack, and a few smaller packages to create our frontend which is served to user's browsers. JavaScript is a mostly single-threaded, nonblocking, asynchronous language.



Frontend - User interface

JavaScript is single threaded, meaning it has one call stack and one memory heap. Due to this fact, it only processes one operation at a time, however, JavaScript supports Web Workers that are separate threads which allow parallel processing without affecting the main thread's responsiveness. JavaScript uses an event driven programming model with an event loop, which continuously checks the message queue for new events or tasks (spinlock). If there are tasks in the queue, the event loop will execute them one by one. The frontend requests resources from the backend and continues to execute while it waits for a response. If it gets a response, it temporarily pauses the current task and processes the response before continuing execution.

The frontend makes requests to the backend using Axios, which builds our HTTP request given the supplied parameters. The response is read back with a response code (403 for forbidden endpoints, 200 for OK, etc) and the body of the response will contain all the request information or an error response if an error occurred.

Performance Comparisons

For our project we decided to compare the performance of three operating systems: Linux (Ubuntu), Windows, and MacOS. The application itself was intended to be run on a Raspberry Pi, but we discovered late into production that our 32-bit Raspberry Pi was unable to run the version of Java that is required to run our application. The operating system primarily used while developing the backend was Linux on Windows using WSL (Windows Subsystem for Linux) running Ubuntu, and this is what we used for our production, testing, and hosting of the server during the presentation (as well as another server run on MacOS for the presentation). We ran into several unexpected issues during production as we were developing the application on three different operating systems. For one, we realized that Windows uses a "\" for directory and file paths as opposed to Linux and MacOS using "/". Additionally, the frontend had to be built using webpack on each system as there were line separator differences between Windows and Mac/Linux.

Process Start Times			Login Times		
Windows	Linux	MacOS	Windows	Linux	MacOS
4.2s	**3.2s**	**2.4s**	**124ms**	**73ms**	**130ms**
Download Times			**Upload Times**		
Windows	Linux	MacOS	Windows	Linux	MacOS
151ms	**71ms**	**105ms**	**300ms**	**144ms**	**185ms**

Comparison times between operating systems. Each machine uploaded/downloaded the same file on the same network. The supplied metrics are an average from many trials.

As pictured above, Linux generally performed much better than MacOS and Windows outside of process start times despite being run as a virtual machine. Simple tasks like logging in took around the same time between Windows and MacOS, but complex actions such as downloading/uploading files took much longer for Windows to complete.

Conclusion

We are very pleased with the final result of our project. Though it ended up being much more complex and difficult than we initially expected, that only makes us all the more proud of how it turned out. We are especially happy with the user interface, neat structure, usability, learning curve, and functionality of the project. This project required all of us to learn concepts that were very foreign to us initially, but we all worked very hard to not only learn but also implement everything we learned to create something great.

Possible Improvements

Given more time, we wanted to make SkyBin a distributed system to further apply class concepts. In this distributed system each server would act as a node with references to all other server nodes. All POST/DELETE/PUT requests would be forwarded to each other node after the lock to do so is obtained. The user would first connect to the primary server which would connect the user to the node with the least number of users so that server loads are balanced.

Pros	Cons
Faster GET request response times	Increase in POST/DELETE/PUT response times
Nodes have a much lower risk of being overloaded via load balancing	Excess computation (replication)
Much higher fault tolerance	Communication overhead

Above we've laid out some of the pros and cons to this approach, though the greatest benefit of creating a distributed system would be to allow users to still access their files in case any server goes down (increase in fault tolerance). This would be favorable as opposed to a single server going down causing no user to be able to access their files.

Despite our Windows server having issues just before our presentation, our presentation went very well. We felt like we covered everything we wanted to and the live demo went perfectly according to plan. Additionally, we were able to have the server running on Linux and MacOS for students to connect to and test themselves for our presentation.

Overall, our presentation not only deepened our understanding developing software, but also supplied us with practical practice for the class itself. Our implementation of multithreading, thread locks, resource management, concurrency, and other class concepts greatly supplemented our learning of these concepts.

Project Contribution

Each week we typically met Monday, Wednesday, and Friday after class for 1-2 hours. We also worked on the project independently for a total average of 3.5 - 4 hours a week (excluding outlier week 10).

Each member's contribution to the project was about equal, with each members' contribution below:

Week	Joshua G Frontend development, architecture design, testing Time Spent approx.	Josh F Database design, frontend development, testing Time Spent approx.	Michael S Database implementation, backend development, testing Time Spent approx.
Week 1: Project Kickoff and Planning Hold a meeting to discuss project requirements and goals. Define roles and responsibilities for each team member. Create a project plan, including milestones and deadlines. Set up communication channels and version control (Discord, GitHub, etc)	4 hrs	3 hrs	5 hrs
Week 2: System Architecture and Design Define the database schema and data models. Outline the RESTful API endpoints for the Java backend. Start working on wireframes for the user interface.	5.5 hrs	6 hrs	5.5 hrs
Week 3: Backend Development - Part 1	3 hrs	5 hrs	5 hrs

Start developing the backend using Java. Implement user authentication and authorization (Becrypt). Begin working on the MySQL database integration. Develop API endpoints for user registration and login.			
Week 4: Backend Development - Part 2 Continue developing backend functionality. Implement file upload and download capabilities. Work on encrypting user data and communication over HTTPS. Test API endpoints using tools like Postman.	4 hrs	4 hrs	4 hrs
Week 5: Frontend Development - Part 1 Begin developing the user-friendly web interface using React.js. Create the user login and registration pages. Set up routing within the React application. Integrate API calls to the backend for user authentication.	5.5 hrs	4 hrs	4 hrs
Week 6: Frontend Development - Part 2 Continue frontend development. Implement file management features (upload, download, delete). Design and implement the dashboard for users. Begin testing the frontend components.	5 hrs	4 hrs	5 hrs
Week 7: Integration and Testing	3.5 hrs	4.5 hrs	3.5 hrs

Integrate the frontend and backend components. Conduct extensive testing to identify and fix any bugs. Perform security testing and ensure data encryption is working correctly. Review and refine the user interface for a polished look.			
Week 8: Documentation and Presentation Preparation Document the project's codebase, APIs, and database schema. Prepare a presentation for the project. Practice the presentation with the team. Finalize any remaining issues and perform a thorough review.	4 hrs	4 hrs	5.5 hrs
Week 9: User Acceptance Testing Conduct user acceptance testing. Gather feedback and make necessary adjustments. Perform performance testing to ensure scalability. Address any last-minute issues or concerns.	4 hrs (decided to do acceptance testing during presentation)	3.5 hrs	3 hrs
Week 10: Final Refinements and Deployment Make final refinements based on user feedback. Prepare the system for deployment to a production environment. Configure the server for HTTPS and ensure secure deployment. Perform one last round of testing in the production environment.	15 hrs (Decided to do in-depth testing using group members since we decided to to acceptance testing during presentation) (Found and fixed A LOT of bugs)	14 hrs	15 hrs

Week 11: Presentation and Project Wrap-Up Deliver the project presentation to the class. Submit all project documentation and code to the instructor. Celebrate the successful completion of the project.	Presentation: Delivered Final Report: In progress	In progress	In progress
Est. Total Time	53.5 hrs	52 hrs	55.5 hrs

References

REST API
https://www.educative.io/answers/restful-web-services
Spring Boot
https://spring.io/guides/gs/spring-boot/
Spring Security
https://spring.io/guides/gs/securing-web/'
Spring Security JWT Authentication
https://thedailytechdose.com/how-to-secure-spring-boot-application-with-jwt/

Appendix A: Project Code

- DirectoryWrapper.java
- FileInfo.java
- FileNode.java
- FolderInfo.java
- SharedFiles.java
- SharedFilesId.java
- UserInfo.java

Authentication Filter

- JwtAuthFilter.java

Configurations

- SecurityConfig.java
- ServerProperties.java
- UserInfoUserDetails.java
- UserInfoUserDetailsService.java

Frontend

Utility Functions

Proxy.js (Utility class for Axios to prevent crashes)

```
/**
 * @type {ProxyHandler}
 */
const handler = {
    get: function (target, property) {
      if (typeof target[property] === "function") {
        return async function (...args) {
          try {
            return await Reflect.apply(target[property], target, args);
          } catch (error) {
            console.log({status:error?.response?.status||null, data:error?.response?.data||null, error})
            return {status:error?.response?.status||null, data:error?.response?.data||null, error};
          }
        };
      } else {
        return Reflect.get(target, property);
      }
    },
  };

export function AxiosProxy(target) { return new Proxy(target, handler) };
```

ServerConnector.js (Utility functions to send & receive data)

```
import axios, { AxiosHeaders } from "axios";
import { AxiosProxy } from './Proxy';

const Axios = AxiosProxy(axios);
window.Axios = Axios;

async function checkLock(){
    let val;
    if (!localStorage.getItem('token')){
```

```
        val = await new Promise(resolve => setTimeout(resolve, 500));
        return await checkLock(!val)
    }else{
        return true;
    }
}

/**
 * Creates the bearer token header
 * @param {AxiosHeaders} otherHeaderInfo
 * @returns
 */
const createAuthHeader = async otherHeaderInfo => ({
    checkLock: await checkLock(),
    headers: { ...(otherHeaderInfo || {}), Authorization: `Bearer ${localStorage.getItem('token')}`, }
});

//checks to see if user is already logged in
export async function loginStillValid() {
    if (localStorage.getItem('token')) {

        let { status, data } = await Axios.get("/api/refresh", await createAuthHeader(), localStorage.getItem('token'));
        if (status === 200) {
            localStorage.setItem('token', data)
            autoRefreshToken()
            return true;
        }

        return false
    }
}

let timeoutID = null;
const TOKEN_REFRESH_TIME = 1000 * 60 * 60; // 1 hour (token expires after 2)
function autoRefreshToken(time = TOKEN_REFRESH_TIME) {
    clearTimeout(timeoutID) // in case of duplicate refresh

    timeoutID = setTimeout(async () => {
        let { status, data } = await Axios.get("/api/refresh", await createAuthHeader(), localStorage.getItem('token'))
        if (status === 200) {
            localStorage.setItem('token', data)
            autoRefreshToken(time)
        }
        else {
            localStorage.removeItem('token')
            location.reload();
        }
    }, time)

}

/**
 * Create an account and login
 * @param {String} username
 * @param {String} password
 * @returns {{success:boolean, response:String}}
 */
export async function createUser(username, password) {

    let { status, data } = await Axios.post("/api/create", { username, password })

    if (status !== 200) return { success: false, response: data };

    return await login(username, password);
}

/**
 * Login to the backend. saves token to session storage
 * @param {String} username
 * @param {String} password
```

```
 * @returns {{success:boolean, response:String}}
 */
export async function login(username, password) {
    let currentTime = Date.now();

    let { data, status } = await Axios.post("/api/authenticate", { username, password })

    console.log(Date.now() - currentTime + " ms taken for login")

    if (status === 200) {
        localStorage.setItem('token', data)
        setTimeout(()=>autoRefreshToken(), 5000)
        return { success: true, response: data };
    }

    return { success: false, response: data };
}

/**
 * Deletes the user account
 * @returns {{success:boolean, response:String}}
 */
export async function deleteAccount() {

    let { data, status } = await Axios.delete("/api/delete", await createAuthHeader(), {})

    return { success: true, response: data };
}

/**
 * gets all files in the users directory
 * @returns {{success:boolean, response:String}}
 */
export async function getDirectory() {
    let { data, status } = await Axios.get("/api/user/listFiles", await createAuthHeader(), {loggedIn: await checkLock()});
    return { success: status===200, response: data };
}

/**
 * uploads a file to the server
 * @param {String} directory
 * @param {File} file
 * @returns {{success:boolean, response:String}}
 */
export async function uploadFile(directory, file){
    let currentTime = Date.now();

    const formData = new FormData();
    formData.append('file', file);
    formData.append('directory', directory);

    let { data, status } = await Axios.post("/api/user/upload", formData, {...(await createAuthHeader()), body:{loggedIn:await
checkLock()},})

    console.log(Date.now() - currentTime + " ms taken for upload");

    return { success: status===200, response: data };
}

/**
 * deletes a file in the server
 * @param {String} directory
 * @param {File} file
 * @returns {{success:boolean, response:String}}
 */
export async function deleteFile(name, directory){
    let { data, status } = await Axios.delete("/api/user/deleteFile", {data:{name, directory}, ...(await createAuthHeader())})
    return { success: status===200, response: data };
}
```

```
/**
 * deletes a folder in the server
 * @param {String} directory
 * @param {File} file
 * @returns {{success:boolean, response:String}}
 */
export async function deleteFolder(name, directory){
    let { data, status } = await Axios.delete("/api/user/deleteFolder", {data:{name, directory}, ...(await createAuthHeader())})
    return { success: status===200, response: data };
}

/**
 * downloads a file
 * @param {String} directory
 * @param {File} file
 * @returns {{success:boolean, response:String}}
 */
export async function downloadFile(filename, directory) {
    let currentTime = Date.now();
    const response = await
fetch(`/api/user/download?filename=${encodeURIComponent(filename)}&directory=${encodeURIComponent(directory)}&loggedIn=${encodeUR
IComponent(await checkLock())}`, {
        headers: (await createAuthHeader()).headers,
        method: 'GET',
        credentials: 'include'
    });

    if (response.ok) {
        try {
            const reader = response.body.getReader();
            const stream = new ReadableStream({
                start(controller) {
                    function push() {
                        reader.read().then(({ done, value }) => {
                            if (done) {
                                controller.close();
                                return;
                            }
                            controller.enqueue(value);
                            push();
                        });
                    }
                    push();
                }
            });

            const blob = await new Response(stream).blob();
            const link = document.createElement('a');
            link.href = URL.createObjectURL(blob);
            link.download = filename;
            document.body.appendChild(link);
            link.click();
            document.body.removeChild(link);
        } catch (e) {
            console.log("internal downloading error")
            return { success: false, response: "Internal Failure, Browser Incompatible" };
        }
    }

    console.log(Date.now() - currentTime + " ms taken for download")

    return { success: response.ok, response: response.statusText };
}

/**
 * Creates a new folder in the specified directory
 * @param {String} name
 * @param {String} directory
 * @returns {{success:boolean, response:String}}
 */
```

```
export async function createFolder(name, directory){
    let { data, status } = await Axios.post("/api/user/newfolder", {name, directory, loggedIn:await checkLock()}, await
createAuthHeader());
    return { success: status===200, response: data };
```

React Entry Point (App.jsx)

```
import React from 'react';
import { createRoot } from 'react-dom/client';
import { BrowserRouter as Router, Routes, Route, useNavigate } from 'react-router-dom';

import './App.scss'
import Login from './Components/User/LoginPopUp/LoginPopUp';
import Sidebar from './Components/Sidebar/Sidebar';

import Directory from './Components/Content Pages/Directory/Directory';
import About from './Components/Content Pages/About/About';
import AccountSettings from './Components/AccountSettings/AccountSettings';

if (document.location.pathname !== "/") document.location.pathname = "/";

function App() {
    return <Router>
        <Sidebar />
        <Routes>
            <Route path='/directory' Component={Directory}></Route>
            <Route path='/about' Component={About}></Route>
            <Route path='/account' Component={AccountSettings}></Route>
            <Route path='*' Component={Login} />
        </Routes>
    </Router>
}

let div = document.body.appendChild(document.createElement('div'));
div.className = 'root';
createRoot(div).render(<App />);
```

App.scss (Formatting and styling of GUI)

```
/* Define variables for reusability */
$primary-color: #007bff;
$secondary-color: #333;

$background: #121735;
$text-color: #41a38a;

* {
    display: block;
    margin: 0;
    padding: 0;

    head,
    style {
        display: none;
    }
}

html, body{
    overflow: hidden;
    width: 100vw;
    height: 100vh;
    background-color: $background;
}

body{
    padding-top: env(safe-area-inset-top);
    padding-bottom: env(safe-area-inset-bottom);
}
```

```
body > * {
    height: 100%;
    overflow: hide;
    -webkit-overflow-scrolling: touch;
}

.root {
    display: flex;
    width: 100%;
    height: 100%;
    overflow: hidden;
}
```

LoginPopUp.jsx (Login component)

```
import React, { useState, useRef, useEffect } from "react";
import { useNavigate } from "react-router-dom";
import { login, createUser, loginStillValid } from "../../../Util/ServerConnector";
import './LoginPopUp.scss'

const numberLines = 10; // if more than 10, change the scss to allow more to render

export default function Login() {
    const navigate = useNavigate(null);

    const [errorState, setErrorState] = useState(false);

    useEffect(()=>{
        loginStillValid().then(bool=>{
            if (bool === true){
                navigate("/directory")
            }
        })
    })

    const hasError = (response="An Error Occured", clearFields=true) => {
        if(clearFields){
            usernameInputRef.current.value = "";
            passwordInputRef.current.value = "";
            usernameInputRef.current.focus();
        }
        if (typeof response === "object")
            response = JSON.stringify(response)
        setErrorState(response);
    }

    const handleLogin = async () => {
        let { success, response } = await login(usernameInputRef.current.value, passwordInputRef.current.value)
        localStorage.setItem("username", usernameInputRef.current.value)
        if (success)
            navigate("/directory")
        else
            hasError(response)
    }

    const createAccount = async () => {
        let { success, response } = await createUser(usernameInputRef.current.value, passwordInputRef.current.value);
        localStorage.setItem("username", usernameInputRef.current.value)
        if (success)
            navigate("/directory")
        else
            hasError(response, false)
    }

    const usernameInputRef = useRef(null); // Ref for the username input
    const passwordInputRef = useRef(null); // Ref for the password input
```

```
    const handleSubmit = (e) => {
        e.preventDefault(); // Prevent the form from submitting
    }

    const checkEnterKey = (e) => {
        if (e.key === 'Enter') {
            e.preventDefault();
            if (usernameInputRef.current === document.activeElement)
                passwordInputRef.current.focus()
            else if (passwordInputRef.current === document.activeElement)
                handleLogin();
        }
    }

    return (
        <div id="LoginPopUp">
            <div id="LoginModal">
                <span id="loginText">Login</span>
                <span id="loginError" style={{display: !errorState ? "none" : "block"}}>{errorState||""}</span>
                <form onSubmit={handleSubmit}>
                    <span id="modalInputs">
                        <input
                            ref={usernameInputRef}
                            type="text"
                            placeholder="username"
                            onKeyDown={checkEnterKey}
                        />
                        <input
                            ref={passwordInputRef}
                            type="password"
                            placeholder="password"
                            onKeyDown={checkEnterKey}
                        />
                    </span>

                    <span id="modalButtons">
                        <button onClick={handleLogin}>
                            Login
                        </button>
                        <button onClick={createAccount}>
                            Create Account
                        </button>
                    </span>

                </form>
            </div>
            <div id="backdrop">
                {[...Array(numberLines).keys()].map(i => <span key={i} className="line" />)}
            </div>
        </div>
    );
}
```

LoginPopUp.scss (GUI styling)

```
$accent: #41a38a;
$background: #121735;
$background-dark: #040717;

$numberLines: 10;

#LoginPopUp {
    position: absolute;
    top: 0;
    left: 0;
    width: 100vw;
    height: 100vh;
    background-color: $background;
    z-index: 999;
```

```scss
display: flex;
justify-content: center;
align-items: center;

@keyframes drop {
    0% {
        top: -50%;
    }

    100% {
        top: 110%;
    }
}

#backdrop {
    z-index: 998;
    position: absolute;
    top: 0vh;
    bottom: 0vh;
    left: 0;
    right: 0;
    display: flex;
    justify-content: space-evenly;
    background-color: rgba($background, 0.5);
    opacity: 0.95;
    backdrop-filter: blur(10px);

    @for $i from 1 through $numberLines {
        $randomNumber: random(5);

        .line:nth-child(#{$i}) {
            position: relative;
            width: 1px;
            height: 100%;
            background: rgba(255, 255, 255, 0.1);

            &::after, &::before {
                content: '';
                display: block;
                position: absolute;
                height: 15vh;
                width: 3px;
                transform: translateX(-1px);
                top: -50%;
                border-radius: 10px;
                background: linear-gradient(to bottom, rgba(255, 255, 255, 0) 0%, #ffffff 75%, #ffffff 100%);
                animation: drop 3s+(random(20)*0.25s) 0s infinite forwards;
                animation-timing-function: cubic-bezier(0.4, 0.26, 0, 0.97);
            }
            &::before{
                animation: drop 2s+(random(20)*0.25s) 0s infinite forwards;
            }
        }
    }
}

#LoginModal{
    background-color: rgba($background-dark, 0.9);
    z-index: 999;
    padding: 20px;
    border-radius: 20px;
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;

    font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida Sans', Arial, sans-serif;
    color: $accent;
```

```
        #loginText{
            font-size: 2em;
            font-weight: bold;
        }

        #loginError{
            font-size: 0.8em;
            color: red;
        }

        form{
            display: flex;
            flex-direction: column;
            justify-content: center;
            align-items: center;
            margin: 10px;

            #modalInputs{
                input{
                    width: 100%;
                    padding: 12px 20px;
                    margin: 8px 0;
                    box-sizing: border-box;
                    border-radius: 10px;
                    opacity: 0.7;
                    border: none;

                    &:focus{
                        opacity: 1;
                    }
                }
            }

            #modalButtons{
                width: 100%;
                display: flex;
                justify-content: space-between;

                button{
                    border: none;
                    padding: 7px;
                    border-radius: 5px;
                    background-color: rgba($accent, 0.8);
                    color: black;
                }
            }
        }
    }
}
}
```

Sidebar.jsx (The sidebar you see on the left side of the app)

```
import React, { useRef } from "react";
import { Link } from "react-router-dom";

import FolderIcon from './Icons/FolderIcon';
import FriendIcon from './Icons/FriendIcon';
import InfoIcon from './Icons/InfoIcon';
import AccountIcon from "./Icons/AccountIcon";

import SkyBinLogo from './SkyBin_logo.png'

import './Sidebar.scss'

export default function Sidebar() {

    const dir = useRef(null);
    const about = useRef(null)
```

```
    const settings = useRef(null)

    let current = null;
    function swap(ref){
        if (current) current.current.classList.remove("active");
        current = ref;
        ref.current.classList.add("active")
    }

    return <nav className='sidebar'>
        <span className='sidebar-top'>
            <img src={SkyBinLogo} alt="Logo" />
        </span>
        <div className='navbar'>
            <Link to="/directory" ref={dir} onClick={()=>swap(dir)}><FolderIcon /></Link>
            <Link to="/about" ref={about} onClick={()=>swap(about)}><InfoIcon /></Link>
            <Link to="/account" ref={settings} onClick={()=>swap(settings)}><AccountIcon /></Link>
        </div>
    </nav>

}
```

Sidebar.scss (Styling for the sidebar)

```
$sidebar-top: #040717;
$sidebar: #121735;
$sidebar-icons: #343a5c;

$button-accent: #41a38a;

.sidebar {
    min-width: 60px;
    max-width: 80px;
    background-color: $sidebar;
    color: white;
    height: 100vh;
    display: flex;
    flex-direction: column;
    box-shadow: 2px 1px 5px 5px black inset;
    height: 100svh;


    .sidebar-top {
        height: 20vh;
        background-color: $sidebar-top;
        margin-bottom: 10vh;
        border-radius: 0 0 10px 10px;
        display: flex;
        flex-direction: column;
        justify-content: space-around;
        overflow: hidden;
        user-select: none;

        img {
            width: 100%;
            border-radius: 100px;
            user-select: none;
        }
    }

    .navbar {
        display: flex;
        flex-direction: column;
        flex: 1;
        height: 100svh;
        overflow-y: auto;


        >* {
```

```
            flex: 1;
            cursor: pointer;
            position: relative;

            svg {
                margin-left: auto;
                margin-right: auto;
            }

            svg path {
                transition: fill 0.5s ease;
                fill: $sidebar-icons;
            }

            width: 70%;
            padding-left: 15%;
            padding-right: 15%;

            &:hover,
            &.active {
                svg path {
                    fill: $button-accent;
                }

                &::after {
                    background-color: $button-accent;
                }
            }

            &::after {
                content: "";
                transition: background-color 0.15s ease;
                position: absolute;
                top: 0;
                right: 0;
                border-radius: 10px;
                height: 100%;
                width: 5%;
            }
        }
    }
}
```

Directory.jsx (The file directory)

```
import React, { useState, useEffect, useRef } from 'react';

import { uploadFile, deleteFile, downloadFile, getDirectory, createFolder, deleteFolder } from "../../../Util/ServerConnector";

import './Directory.scss'

import DirectoryIcon from '../../../FileSystemIcons/DirectoryIcon'

import CreateFolder from '../../../FileSystemIcons/menubarIcons/create-folder.svg';
import DeleteItem from '../../../FileSystemIcons/menubarIcons/delete.svg';
import Download from '../../../FileSystemIcons/menubarIcons/download.svg'
import Upload from '../../../FileSystemIcons/menubarIcons/upload.svg'
import Back from '../../../FileSystemIcons/menubarIcons/back.svg'

export default function Directory() {

    const [data, setData] = useState({ name: "", path: null, contents: [], isDirectory: true });
    const [currentPath, setPath] = useState("");

    useEffect(() => {
        getDirectory().then(({ response, success }) => {
            if (success) {
                localStorage.setItem('username', response.name)
                setData(normalize(response));
```

```
            setPath(response.name);
        }
    })
}, []);

function normalize(currentObj) {
    currentObj.path = currentObj?.path?.replaceAll("/", "\\") || "\\";

    if (currentObj.contents && currentObj.contents.length > 0) {
        currentObj.contents.map((el) => normalize(el))
    }

    return currentObj
}

function isValidName(string) {
    // Define the regular expression
    const regex = /^([a-zA-Z0-9]| )+$/;

    // Use the test method to check if the input matches the regex
    return regex.test(string);
}

let currentSelected = null;
function changeSelected(id) {
    if (currentSelected)
        currentSelected.classList.remove("selected")

    if (id === null) {
        currentSelected = null
        return
    };
    currentSelected = document.querySelector("#" + id);
    currentSelected.classList.add("selected");
}

const fileInputRef = useRef(null);
const handleSvgClick = () => {
    fileInputRef.current.click();
};
const handleFileChange = async (event) => {
    if (event.target.files[0]) {
        let directory;
        if (currentPath.indexOf("\\") === -1)
            directory = "\\"
        else
            directory = currentPath.substring(currentPath.indexOf("\\") + 1, currentPath.length)

        let { success, response } = await uploadFile(directory, event.target.files[0]);
        if (!success)
            alert(response || "File size exceeded")
        refreshFiles()
    }

};

const handleDownload = () => {
    if (!currentSelected) return;
    let data = JSON.parse(currentSelected.getAttribute("data-file"));
    if (data.isDirectory) return;
    downloadFile(data.name, currentPath.substring(currentPath.indexOf("\\"), currentPath.length))
    changeSelected(null)
}

const FolderInputRef = useRef(null);
const FolderDivRef = useRef(null);
function handleClickOutsideBox(event, override = false) {
    if (override || !FolderDivRef.current.contains(event.target)) {
        document.removeEventListener('click', handleClickOutsideBox);
        FolderDivRef.current.classList.add("hidden")
```

```
            FolderInputRef.current.value = "";
        }
    }
    function HandleCreateFolder() {

        FolderDivRef.current.classList.remove("hidden");
        setTimeout(() => document.addEventListener('click', handleClickOutsideBox), 100);
    }
    function HandleCreateFolderConfirmation() {
        let name = FolderInputRef.current.value;
        if (!name || name === "") return;
        if (!isValidName(name)) {
            alert("Please do not use special characters");
            return;
        }

        let directory;
        if (currentPath?.indexOf("\\") === -1)
            directory = "\\"
        else
            directory = currentPath.substring(currentPath.indexOf("\\") + 1, currentPath.length)
        handleClickOutsideBox(null, true)

        createFolder(name, directory).then(({ success, response }) => {
            if (!success)
                alert(response || "Something Happened Wrongly")
            else
                refreshFiles()
        })

    }

    function HandleFolderChange(name, goingback = false, event = null) {
        if (event) event.stopPropagation();
        changeSelected(null)

        if (!goingback) {
            setPath(currentPath + "\\" + name)
        } else {
            if (currentPath.lastIndexOf("\\") !== -1)
                setPath(currentPath.substring(0, currentPath.lastIndexOf("\\")))
        }
    }

    function handleDelete() {
        if (!currentSelected) return;
        let data = JSON.parse(currentSelected.getAttribute("data-file"));

        let directory;
        if (currentPath.indexOf("\\") === -1)
            directory = "\\"
        else
            directory = currentPath.substring(currentPath.indexOf("\\") + 1, currentPath.length)

        if (data.isDirectory) {
            deleteFolder(data.name, directory).then(({ response, success }) => {
                if (!success)
                    alert(response + " (It could be your folder has contents within it)")
                refreshFiles()
            })
        } else {
            deleteFile(data.name, directory).then(({ response, success }) => {
                //if (!success)
                //alert(response)
                refreshFiles()
            })
        }
    }

    function refreshFiles() {
```

```
        if (currentPath === "" && data.name !== "") setPath(data.name);
        getDirectory().then(({ response, success }) => {
            if (success) {
                localStorage.setItem('username', response.name)
                setData(normalize(response));
            }
        });
        changeSelected(null)
    }

    function getObjectAtPath(currentPath, obj) {
        if (!currentPath) return obj;

        const pathSegments = currentPath.split("\\");
        let currentObj = { contents: [obj] };

        for (const segment of pathSegments) {
            // Find the matching content by name in the current object's contents
            const matchingContent = currentObj.contents.find(content => content.name === segment);

            // If the matching content is found, update the current object
            if (matchingContent) {
                currentObj = matchingContent;
            } else {
                // If not found path does not exist, just return root
                return { ...obj, successfulMove: false };
            }
        }

        return { ...currentObj, successfulMove: true };
    }

    return <div className='content_container'>
        {setTimeout(()=>{refreshFiles()}, currentPath===""?400: 30000)}
        <Back className="undoMove" onClick={() => HandleFolderChange(null, true)} />
        <div className='dirpath'>{currentPath}</div>
        <div className='top_bar'>
            <input style={{ display: "none" }} type='file' ref={fileInputRef} onChange={handleFileChange} />
            <Upload onClick={handleSvgClick} />

            <Download onClick={handleDownload} />

            <div className='CreateFolderPopUp hidden' ref={FolderDivRef} tabIndex={0}>
                <input ref={FolderInputRef} type='text'></input>
                <button onClick={HandleCreateFolderConfirmation}>Save</button>
            </div>
            <CreateFolder onClick={HandleCreateFolder} />

            <DeleteItem onClick={handleDelete} />
        </div>
        <div className='dir_container'>
            {getObjectAtPath(currentPath || localStorage.getItem('username'), data).contents.map(({ name, path, contents,
isDirectory }, i) =>
                <span key={i} id={'fileItem' + i} className='file' onClick={() => { changeSelected('fileItem' + i) }}
data-file={JSON.stringify({ name, path, contents, isDirectory })}>
                    <DirectoryIcon name={name} isDirectory={isDirectory} />
                    <span className='filename'>{name}</span>
                    {isDirectory ? <span className='folderOpen' onClick={(e) => HandleFolderChange(name, null, e)}>Open</span> :
<></>}
                </span>
            )}
        </div>
    </div>
}
```

Directory.scss (GUI Styling)

```
/* Define variables for reusability */
$primary-color: #007bff;
```

```scss
$secondary-color: #333;

$background: #121735;
$text-color: #41a38a;
$file-hover-color: #182b54;

.hidden{
    display: none;
}

.CreateFolderPopUp{
    position: absolute;
    padding: 10px;
    background-color: rgba(black, 0.9);
    backdrop-filter: blur(10px);
    border-radius: 10px;
    width: 50vw;

    input[type=text] {
        box-sizing: border-box;
        width: 100%;
        padding: 12px 5px;
        margin: 8px 0;
        border: none;
        outline: none;
    }

    button{
        box-sizing: border-box;
        width: 100%;
        color: $text-color;
        background-color: $background;
        border: none;

        :focus{
            background-color: black;
        }
    }
}

.content_container {
    width: 100%;
    height: 100vh;
    background-color: $background;

    display: flex;
    flex-direction: column;
    position: relative;

    align-items: center;

    .dirpath{
        display: inline-block;
        position: absolute;
        top: 10vh;
        width: fit-content;
        max-width: 90vw;
        height: 15px;
        padding: 4px;
        fill: rgba($text-color, 0.5);
        z-index: 99;
        background-color: rgba(black, 0.5);
        backdrop-filter: blur(10px);
        border-radius: 10px;
        overflow: hidden;
        text-overflow: ellipsis;
        color: $text-color;
        text-wrap: none;
        white-space: nowrap;
```

```scss
    path{
        fill: inherit;
    }
}

.undoMove{
    position: absolute;
    right: 5%;
    top: 0;
    width: 8vh;
    height: 8vh;
    margin: 1vh;
    fill: rgba($text-color, 0.5);
    z-index: 99;
    background-color: rgba(black, 0.5);
    backdrop-filter: blur(10px);
    border-radius: 50px;

    @media only screen and (max-width: 500px) {
        top: calc(10vh + 15px);
    }

    path{
        fill: inherit;
    }
}

.top_bar {
    backdrop-filter: blur(10px);
    height: 8vh;
    position: absolute;
    z-index: 99;
    top: 0;
    color: white;
    background-color: rgba(black, 0.5);
    margin: 1vh;
    border-radius: 10px;
    min-width: 50%;
    display: flex;

    svg {
        width: 25%;
        height: 100%;
        fill: rgba($text-color, 0.5);
        transition: background-color 0.25s ease-out;
        transition: fill 0.25s ease-out;

        path {
            fill: inherit;
        }

        &:hover{
            fill: white;
            background-color: rgba(white, 0.1);
            border-radius: 10px;
        }
    }
}

.dir_container {
    background-color: $background;

    padding-top: 10vh;
    padding-bottom: 5vh;

    display: flex;
    flex-wrap: wrap;
    overflow-y: auto;
    gap: 10px;
    justify-content: center;
```

```scss
.file {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    position: relative;
    width: 200px;
    min-width: 90px;

    &:not(:hover),
    .selected {


        svg {
            fill: rgba($text-color, 0.7);

            path {
                fill: rgba($text-color, 0.7);
            }
        }

        .filename {
            opacity: 1;
        }
    }

    &:hover,
    &.selected {
        cursor: pointer;
        background-color: rgba(black, 0.5);
        border-radius: 10px;
    }

    &.selected {
        border: white solid 1px;

        align-items: center;
        justify-content: center;

        .folderOpen{
            user-select: none;
            display: flex;
            position: absolute;
            color: white;
            padding: 5% 8%;
            background-color: $background;
            border-radius: 50px;
        }
    }

    .folderOpen{
        display: none;
    }

    .filename {
        text-align: center;
        font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
        font-size: 100%;
        color: $text-color;
        font-weight: bold;
        position: absolute;
        bottom: 0;
        width: 100%;
        margin-left: auto;
        margin-right: auto;
        overflow: hidden;
        white-space: nowrap;
        opacity: 0.7;
    }
```

```
            svg {

                transition: fill 0.5s ease-in;
                //fill: $background;

                path {
                    transition: fill 0.25s ease-out;
                    //fill: $background;
                }
            }
        }

    }
}
```

About.jsx (About page)

```
import './About.scss'

import React from 'react';

export default function About() {

    return <div id="aboutPage">
        <h1 className='aboutTitle'>About</h1>
        <section className='aboutBody'>
            <p>SkyBin is a Network Attached Storage (NAS) application accessible over HTTP, incorporating user authentication and authorization features. Users have the ability to upload, download, and manage files in our file database. The user interface will be designed using JavaScript and React.js, while the backend is implemented with Java and SpringBoot. User information, such as usernames and passwords, is stored in a MySQL database. This project covers class concepts such as multithreading, child process/thread management, and exception handling.</p>
        </section>
    </div>
}
```

About.scss (GUI Styling)

```
$primary-color: #007bff;
$secondary-color: #333;

$background: #121735;
$text-color: #41a38a;
$file-hover-color: #182b54;

#aboutPage{
    display: flex;
    flex-direction: column;
    align-items: center;
    text-align: center;
    background-color: $background;
    width: 100%;
    height: 100%;

    display: flex;

    .aboutTitle{
        color: $text-color;
        margin: 2%;
    }

    .aboutBody{
        color: white;
        margin: 5% 20%;
        font-size: 1.2em;
    }
}
```

AccountSettings.jsx (Page for logging out and deleting account)

```jsx
import { deleteAccount } from '../../Util/ServerConnector';
import './AccountSettings.scss'

import React from 'react';

export default function AccountSettings(){

    function LogOut(){
        localStorage.clear();
        document.location.pathname = '/';
    }

    function deleteAcc(){
        deleteAccount().then(LogOut);
    }

    return <div className="AccountSettings">
        <div className='accountItem'>
        <h1 className='ItemTitle'>Settings</h1>
            <button onClick={LogOut}>Log Out</button>
            <button onClick={deleteAcc} style={{color:"red"}}>Delete Account</button>
        </div>
    </div>
}
```

AccountSettings.scss (GUI styling)

```scss
/* Define variables for reusability */
$primary-color: #007bff;
$secondary-color: #333;

$background: #121735;
$text-color: #41a38a;
$file-hover-color: #182b54;

.AccountSettings {
    display: flex;
    width: 100%;
    flex-direction: column;
    color: $text-color;
    background-color: $background;
    padding: 12px;

    .accountItem {
        display: flex;
        flex-direction: column;
        padding: 2%;
        height: 50vh;
        align-items: center;

        .ItemTitle {
            margin-bottom: 100px;
        }

        button {
            box-sizing: border-box;
            color: $text-color;
            background-color: rgba(black, 0.7);
            border: none;
            border-radius: 10px;
            padding: 8px;
            margin: 2px;
            width: 100px;
            transition: all 0.3s;

            &:hover {
                background-color: rgba(white, 0.1);
```

```
            }
        }
    }

}
```

webpack.config.js (Build and development configuration)

```
const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');
const { CleanWebpackPlugin } = require('clean-webpack-plugin');
const MiniCssExtractPlugin = require('mini-css-extract-plugin');
const TerserPlugin = require('terser-webpack-plugin');

const webpack = require('webpack')

module.exports = (env, argv) => {
    const isDevelopment = argv.mode === 'development';
    console.log("Development: " + isDevelopment);

    return {
        devtool: isDevelopment ? "eval" : "source-map",
        entry: './src/App.jsx',
        output: {
            path: path.resolve(__dirname, 'dist'),
            filename: isDevelopment ? '[name].js' : '[name].[contenthash].js',
            publicPath: '/',
            clean: true,
        },
        module: {
            rules: [
                {
                    test: /\.(js|jsx)$/,
                    exclude: /node_modules/,
                    use: "babel-loader"
                },
                {
                    test: /\.(css|scss)$/,
                    exclude: /node_modules/,
                    use: [
                        isDevelopment ? 'style-loader' : MiniCssExtractPlugin.loader,
                        'css-loader',
                        'sass-loader'
                    ]
                },
                {
                    test: /\.(jpe?g|png|gif)$/i,
                    type: 'asset/resource'
                },
                {
                    test: /\.svg$/,
                    use: ['@svgr/webpack'],
                }
            ]
        },
        resolve: {
            extensions: ['.js', '.jsx', '.ts', '.tsx', '.css', '.scss', '.sass', '']
        },
        plugins: [
            new webpack.DefinePlugin({
                "process.env.isDev": isDevelopment
            }),
            new CleanWebpackPlugin(), // Clean builds not in use
            new MiniCssExtractPlugin({
                filename: isDevelopment ? 'main.css' : 'main.[contenthash].css',
            }),
            new HtmlWebpackPlugin({
                title: "SkyBin"
            })
```

```
        ],
        optimization: {
            moduleIds: 'deterministic',
            runtimeChunk: 'single',
            splitChunks: {
                cacheGroups: {
                    vendor: {
                        test: /[\\/]node_modules[\\/]/,
                        name: 'vendors',
                        chunks: 'all',
                    },
                },
            },
            minimizer: [new TerserPlugin()],
        },
        devServer: {
            static: {
                directory: path.join(__dirname, 'public'),
            },
            compress: true,
            port: 5050,
            proxy: {
                '/api': {
                    target: 'http://localhost:4000', // The URL of the external API server
                    changeOrigin: true,
                }
            }
        }
    }
}
```

Backend Code

SkybinApplication.java (Entry File)

```
package proj.skybin;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SkybinApplication {

    public static void main(String[] args) {
        SpringApplication.run(SkybinApplication.class, args);
    }

}
```

Services

FileService.java

```
package proj.skybin.service;

import proj.skybin.model.FileInfo;
import proj.skybin.model.FolderInfo;

import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.List;
import java.util.Optional;
```

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import jakarta.transaction.Transactional;
import proj.skybin.repository.FileRepository;
import proj.skybin.repository.FolderRepository;

@Service
public class FileService {

    @Autowired
    private FileRepository fileRepository;

    @Autowired
    private FolderRepository folderRepository;

    public FileInfo createFile(FileInfo f) {
        String pathString = "";
        String parentPathString = "";
        // remove everything in the path after the owner's name
        String[] path = f.getPath().split(f.getOwner());
        if (path.length == 1) {
            pathString = f.getOwner();
            f.setPath(pathString);
        } else {
            pathString = f.getOwner();
            for (int i = 1; i < path.length; i++) {
                pathString = pathString + path[i];
            }
            f.setPath(pathString);
        }
        // find parent folder
        Path parentPath = Paths.get(pathString).getParent();
        // if parent folder exists, set parent directory
        if (parentPath != null) {
            parentPathString = parentPath.toString();
            Optional<FolderInfo> parent = folderRepository.findByPath(parentPathString);
            if (parent.isPresent()) {
                f.setParent(parent.get());
                if (parent.get().getFiles() == null) {
                    parent.get().setFiles(new java.util.ArrayList<>());
                }
                parent.get().getFiles().add(f);
                folderRepository.save(parent.get());
                f.setParentpath(parentPath.toString());
            }
        }
        return fileRepository.save(f);
    }

    public Optional<FileInfo> getFile(String filepath) {
        return fileRepository.findByPath(filepath);
    }

    public FileInfo getFile(String owner, String directory, String filename) {
        return fileRepository.findByOwnerAndDirectoryAndName(owner, directory, filename).orElse(null);
    }

    public List<FileInfo> getDirectoryContents(String directory, String owner) {
        return fileRepository.findByDirectoryAndOwner(directory, owner);
    }

    public List<FileInfo> getAllFiles(String owner) {
        return fileRepository.findByOwner(owner);
    }

    @Transactional
    public void deleteFile(String filepath) {
        // remove file from parent folder
```

```
        Path parentPath = Paths.get(filepath).getParent();
        Optional<FolderInfo> parent = folderRepository.findByPath(parentPath.toString());
        if (parent.isPresent()) {
            parent.get().getFiles().removeIf(f -> f.getPath().equals(filepath));
            folderRepository.save(parent.get());
        }
        fileRepository.deleteByPath(filepath);
    }

    // update file name
    // this will update the file's name
    // this will also update the file's path
    public boolean renameFile(String filepath, String newName) {
        Optional<FileInfo> file = fileRepository.findByPath(filepath);
        if (file.isPresent()) {
            // update file name
            file.get().setName(newName);
            // update file path
            String newPath = filepath.substring(0, filepath.lastIndexOf("/") + 1) + newName;
            file.get().setPath(newPath);
            fileRepository.save(file.get());
            return true;
        }
        return false;
    }
}
```

FolderService.java

```
package proj.skybin.service;

import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import proj.skybin.model.FileInfo;
import proj.skybin.model.FolderInfo;
import proj.skybin.repository.FolderRepository;

@Service
public class FolderService {

    @Autowired
    private FolderRepository folderRepository;

    @Autowired
    private FileService fileService;

    public FolderInfo createFolder(FolderInfo folder) {
        // remove everything in the path after the owner's name
        String[] path = folder.getPath().split(folder.getOwner());
        String pathString = "";
        String parentPathString = "";
        if (path.length == 1) {
            pathString = folder.getOwner();
            folder.setPath(pathString);
        } else {
            pathString = folder.getOwner();
            for (int i = 1; i < path.length; i++) {
                pathString = pathString + path[i];
            }
            folder.setPath(pathString);
        }
        // find parent folder
        Path parentPath = Paths.get(pathString).getParent();
```

```
        if (parentPath != null) {
            parentPathString = parentPath.toString();
            Optional<FolderInfo> parent = folderRepository.findByPath(parentPathString);
            if (parent.isPresent()) {
                if (parent.get().getSubfolders() == null) {
                    parent.get().setSubfolders(new ArrayList<>());
                }
                parent.get().getSubfolders().add(folder);
                folder.setParent(parent.get());
                folderRepository.save(parent.get());
            }
        }
        return folderRepository.save(folder);
    }

    public FolderInfo getFolder(String folderpath) {
        return folderRepository.findByPath(folderpath).orElse(null);
    }

    public FolderInfo getFolder(String owner, String directory, String foldername) {
        return folderRepository.findByOwnerAndDirectoryAndName(owner, directory, foldername).orElse(null);
    }

    public List<FolderInfo> getHomeDirectoryContents(String owner) {
        return folderRepository.findByOwner(owner);
    }

    public void deleteFolder(String folderpath) {
        // remove folder from parent folder
        Path parentPath = Paths.get(folderpath).getParent();
        if (parentPath != null) {
            Optional<FolderInfo> parent = folderRepository.findByPath(parentPath.toString());
            if (parent.isPresent()) {
                parent.get().getSubfolders().removeIf(f -> f.getPath().equals(folderpath));
                folderRepository.save(parent.get());
            }
        }
        // delete all subfolders
        FolderInfo folder = folderRepository.findByPath(folderpath).orElse(null);
        if (folder != null) {
            // delete all subfolders
            if (folder.getSubfolders() != null) {
                List<FolderInfo> subfoldersCopy = new ArrayList<>(folder.getSubfolders());
                for (FolderInfo subfolder : subfoldersCopy) {
                    deleteFolder(subfolder.getPath());
                }
                folder.getSubfolders().clear();
            }
            // delete all files
            if (folder.getFiles() != null) {
                List<FileInfo> filesCopy = new ArrayList<>(folder.getFiles());
                for (FileInfo file : filesCopy) {
                    fileService.deleteFile(file.getPath());
                }
                folder.getFiles().clear();
            }
        }
        // delete folder
        folderRepository.deleteById(folderpath);
    }

    public void updateFolder(FolderInfo folder) {
        folderRepository.save(folder);
    }

    public FolderInfo getFolderByPath(String path) {
        return folderRepository.findByPath(path).orElse(null);
    }

    // rename folder
```

```
    // this will rename the folder
    // this will also rename the folder's directory
    // this will also rename the folder's path
    public boolean renameFolder(String folderpath, String newFoldername) {
        FolderInfo folder = folderRepository.findByPath(folderpath).orElse(null);
        if (folder != null) {
            // rename folder
            folder.setName(newFoldername);
            // rename directory
            String[] path = folder.getDirectory().split(folder.getName());
            String directory = "";
            if (path.length == 1) {
                directory = folder.getName();
            } else {
                directory = folder.getName();
                for (int i = 1; i < path.length; i++) {
                    directory = directory + path[i];
                }
            }
            folder.setDirectory(directory);
            // rename path
            String[] path2 = folder.getPath().split(folder.getName());
            String pathString = "";
            if (path2.length == 1) {
                pathString = folder.getName();
            } else {
                pathString = folder.getName();
                for (int i = 1; i < path2.length; i++) {
                    pathString = pathString + path2[i];
                }
            }
            folder.setPath(pathString);
            folderRepository.save(folder);
            return true;
        }
        return false;
    }
}
```

JwtService.java

```
package proj.skybin.service;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.io.Decoders;
import io.jsonwebtoken.security.Keys;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;

import java.security.Key;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;

@Component
public class JwtService {

    public static final String SECRET = {jwtsecret};

    public String extractUsername(String token) {
        return extractClaim(token, Claims::getSubject);
    }
```

```
    public Date extractExpiration(String token) {
        return extractClaim(token, Claims::getExpiration);
    }

    public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
        final Claims claims = extractAllClaims(token);
        return claimsResolver.apply(claims);
    }

    private Claims extractAllClaims(String token) {
        return Jwts
                .parserBuilder()
                .setSigningKey(getSignKey())
                .build()
                .parseClaimsJws(token)
                .getBody();
    }

    private Boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());
    }

    public Boolean validateToken(String token, UserDetails userDetails) {
        final String username = extractUsername(token);
        return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
    }

    public String generateToken(String userName) {
        Map<String, Object> claims = new HashMap<>();
        return createToken(claims, userName);
    }

    private String createToken(Map<String, Object> claims, String userName) {
        return Jwts.builder()
                .setClaims(claims)
                .setSubject(userName)
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 120))
                .signWith(getSignKey(), SignatureAlgorithm.HS256).compact();
    }

    private Key getSignKey() {
        byte[] keyBytes = Decoders.BASE64.decode(SECRET);
        return Keys.hmacShaKeyFor(keyBytes);
    }

}
```

SharedFilesService.java

```
package proj.skybin.service;

import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import proj.skybin.repository.SharedFilesRepository;
import proj.skybin.model.SharedFiles;
import java.util.List;

@Service
public class SharedFilesService {

    @Autowired
    private SharedFilesRepository sharedFilesRepository;
```

```
    public SharedFiles share(SharedFiles sharedFiles) {
        return sharedFilesRepository.save(sharedFiles);
    }

    public List<SharedFiles> getSharedFiles(String sharedUser) {
        return sharedFilesRepository.findBySharedUser(sharedUser);
    }

    public boolean unshare(String path, String owner, String sharedUser) {
        Optional<SharedFiles> sharedFile = sharedFilesRepository.findByPathAndOwnerAndSharedUser(path, owner,
                sharedUser);
        if (sharedFile.isPresent()) {
            sharedFilesRepository.delete(sharedFile.get());
            return true;
        }
        return false;
    }
}
```

UserService.java

```
package proj.skybin.service;

import proj.skybin.model.AuthRequest;
import proj.skybin.model.UserInfo;

public interface UserService {
    UserInfo createUser(UserInfo user);
    String login(AuthRequest login);
    Boolean deleteUser(String username);
    Boolean updateUsername(String username, String newUsername);
    Boolean lock(String username);
    Boolean unlock(String username);
    Boolean userExists(String username);
}
```

UserServiceImpl.java

```
package proj.skybin.service;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import proj.skybin.repository.UserRepository;
import proj.skybin.model.UserInfo;
import proj.skybin.model.AuthRequest;
import proj.skybin.model.FolderInfo;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

@Service
public class UserServiceImpl implements UserService {
    @Autowired
    private UserRepository userRepository;
    @Autowired
    private PasswordEncoder passwordEncoder;
    @Autowired
    private FolderService folderService;
```

```java
@Override
public UserInfo createUser(UserInfo u) {
    if (userRepository.findByUsername(u.getUsername()).isPresent()) {
        return null;
    }
    u.setPassword(passwordEncoder.encode(u.getPassword()));
    userRepository.save(u);
    try {
        Files.createDirectory(Paths.get(System.getProperty("user.dir"), "filedir", u.getUsername()));
        FolderInfo f = new FolderInfo();
        f.setPath(Paths.get(System.getProperty("user.dir"), "filedir", u.getUsername()).toString());
        f.setOwner(u.getUsername());
        f.setDirectory("root");
        f.setName(u.getUsername());
        folderService.createFolder(f);

    } catch (IOException e) {
        System.out.println("Directory already exists");
    }
    return u;
}

@Override
public String login(AuthRequest login) {
    Optional<UserInfo> oUser = userRepository.findByUsername(login.getUsername());
    if (oUser.isPresent()) {
        UserInfo user = oUser.get();
        String encryptedPass = user.getPassword();
        boolean passwordMatch = passwordEncoder.matches(login.getPassword(), user.getPassword());
        if (passwordMatch) {
            Optional<UserInfo> userOptional = userRepository.findByUsernameAndPassword(login.getUsername(),
                    encryptedPass);
            if (userOptional.isPresent()) {
                return "Login successful";
            } else {
                return "Login failed";
            }
        } else {
            return "Passwords do not match";
        }
    } else {
        return "Email not found";
    }
}

@Override
public Boolean deleteUser(String username) {
    Optional<UserInfo> user = userRepository.findByUsername(username);
    if (user.isPresent()) {
        userRepository.delete(user.get());
        // delete user's directory
        folderService.deleteFolder(username);
        return true;
    } else {
        return false;
    }
}

@Override
public Boolean updateUsername(String username, String newUsername) {
    Optional<UserInfo> user = userRepository.findByUsername(username);
    if (user.isPresent()) {
        user.get().setUsername(newUsername);
        userRepository.save(user.get());
        // rename user's directory
        Path path = Paths.get(System.getProperty("user.dir"), "filedir", username);
        Path newPath = Paths.get(System.getProperty("user.dir"), "filedir", newUsername);
        // check if the folder exists
        if (path.toFile().exists()) {
```

```
                // rename the folder
                try {
                    Files.move(path, newPath);
                } catch (IOException e) {
                    System.out.println("Error renaming user's directory");
                }
            }
            return true;
        }
        return false;
    }

    @Override
    public Boolean lock(String username) {
        Optional<UserInfo> user = userRepository.findByUsername(username);
        if (user.isPresent()) {
            while (user.get().getLock()) {
                try {
                    Thread.sleep(1000);
                    user = userRepository.findByUsername(username);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            user.get().setLock(true);
            userRepository.save(user.get());
            return true;
        }
        return false;
    }

    @Override
    public Boolean unlock(String username) {
        Optional<UserInfo> user = userRepository.findByUsername(username);
        if (user.isPresent()) {
            user.get().setLock(false);
            userRepository.save(user.get());
            return true;
        }
        return false;
    }

    // check if user exists
    public Boolean userExists(String username) {
        Optional<UserInfo> user = userRepository.findByUsername(username);
        return user.isPresent();
    }
}
```

Controllers

FileController.java

```
package proj.skybin.controller;

import java.io.IOException;
import java.nio.file.FileVisitResult;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.SimpleFileVisitor;
import java.nio.file.attribute.BasicFileAttributes;
import java.security.Principal;
import java.util.List;
import java.time.Instant;
```

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.io.FileSystemResource;
import org.springframework.core.io.Resource;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.web.server.ResponseStatusException;

import proj.skybin.model.DirectoryWrapper;
import proj.skybin.model.FileInfo;
import proj.skybin.model.FileNode;
import proj.skybin.model.FolderInfo;
import proj.skybin.service.FileService;
import proj.skybin.service.FolderService;
import proj.skybin.service.UserService;

@RestController
@RequestMapping("/api/user")
public class FileController {

    @Autowired
    private FileService fileService;

    @Autowired
    private FolderService folderservice;

    @Autowired
    private UserService userService;

    // for testing the addition of new files to the database, can also be used to
    // create a newfile without an upload (?)
    @PostMapping("/newfile")
    public FileInfo createFile(Principal principal, @RequestBody FileInfo f) {
        userService.lock(principal.getName());
        String owner = principal.getName();
        f.setOwner(owner);
        userService.unlock(principal.getName());
        return fileService.createFile(f);
    }

    // upload a file to the server
    // the file is stored in the fildir folder within the user's home folder
    // and within the directory specified in the request
    // the file information is automatically added to the database
    // uses the provided token to get the username
    @PostMapping("/upload")
    public ResponseEntity<String> uploadFile(Principal principal, @RequestParam String directory,
            @RequestParam MultipartFile file) throws IOException {
        Instant start = Instant.now();
        userService.lock(principal.getName());
        String filename = file.getOriginalFilename();
        // check if the directory is null/empty/invalid
        if (directory == null || directory.equals("null") || directory.equals("") || directory.equals("\\")) {
            directory = "/";
        }
        // check if the file is empty
        if (file.isEmpty()) {
            userService.unlock(principal.getName());
```

```
            return ResponseEntity.badRequest().body("File is empty");
        }
        // check if the file already exists
        Path path = Paths.get(System.getProperty("user.dir"), "filedir", principal.getName(), directory,
                file.getOriginalFilename());
        System.out.println(path.toString());
        if (Files.exists(path)) {
            // add a number to the end of the filename
            int i = 1;
            for (; i < 100; i++) {
                path = Paths.get(System.getProperty("user.dir"), "filedir", principal.getName(), directory,
                        file.getOriginalFilename() + "(" + i + ")");
                if (!Files.exists(path)) {
                    break;
                }
            }
            // change the name of the file
            filename = file.getOriginalFilename() + "(" + i + ")";
            path = Paths.get(System.getProperty("user.dir"), "filedir", principal.getName(), directory, filename);
        }
        // recieve the file and save it to the database
        try {
            Files.copy(file.getInputStream(), path);
        } catch (IOException e) {
            try {
                // replace all back slashes with forward slashes
                String newPath = path.toString().replace("\\", "/");
                System.out.println(newPath);
                path = Paths.get(newPath);
                Files.copy(file.getInputStream(), path);
            } catch (IOException e2) {
                userService.unlock(principal.getName());
                return ResponseEntity.badRequest().body("Failed to upload file");
            }
        }
        FileInfo f = new FileInfo();
        f.setOwner(principal.getName());
        f.setName(filename);
        f.setDirectory(directory);
        f.setPath(path.toString());
        f.setIsDirectory(false);
        // add the file to the database
        fileService.createFile(f);
        userService.unlock(principal.getName());
        Instant end = Instant.now();
        System.out.println("Time to upload: " + (end.toEpochMilli() - start.toEpochMilli()) + "ms");
        return ResponseEntity.ok("File was uploaded successfully");

    }

    // download a file from the server
    // uses the provided token to get the username
    @GetMapping("/download")
    public ResponseEntity<Resource> getFile(Principal principal, @RequestParam String directory,
            @RequestParam String filename) throws IOException {
        System.out.println("Directory: " + directory);
        System.out.println("Filename: " + filename);
        // save current timestamp
        Instant start = Instant.now();
        userService.lock(principal.getName());
        if (directory == null || directory.equals("null") || directory.equals("") || directory.equals("\\")
                || directory.equals(principal.getName())) {
            directory = "/";
        }
        // get the file from the server
        Path pathObject = Paths.get(System.getProperty("user.dir"), "filedir", principal.getName(), directory,
                filename);
        String path = pathObject.toString();
        System.out.println(path);
        Resource resource = new FileSystemResource(path);
```

```
        // check if the file exists
        if (resource.exists()) {
            String mimeType = Files.probeContentType(Paths.get(path + "/" + filename));
            if (mimeType == null) {
                mimeType = "application/octet-stream";
            }
            userService.unlock(principal.getName());
            Instant end = Instant.now();
            System.out.println("Time to download: " + (end.toEpochMilli() - start.toEpochMilli()) + "ms");
            return ResponseEntity.ok()
                    .contentType(MediaType.parseMediaType(mimeType))
                    .header(HttpHeaders.CONTENT_DISPOSITION, "attachment; filename=\"" + resource.getFilename() + "\"")
                    .body(resource);
        } else {
            // replace all back slashes with forward slashes
            path = path.replace("\\", "/");
            resource = new FileSystemResource(path);
            if (resource.exists()) {
                String mimeType = Files.probeContentType(Paths.get(path + "/" + filename));
                if (mimeType == null) {
                    mimeType = "application/octet-stream";
                }
                userService.unlock(principal.getName());
                Instant end = Instant.now();
                System.out.println("Time to download: " + (end.toEpochMilli() - start.toEpochMilli()) + "ms");
                return ResponseEntity.ok()
                        .contentType(MediaType.parseMediaType(mimeType))
                        .header(HttpHeaders.CONTENT_DISPOSITION,
                                "attachment; filename=\"" + resource.getFilename() + "\"")
                        .body(resource);
            } else {
                userService.unlock(principal.getName());
                return ResponseEntity.badRequest().body(null);
            }
        }
    }

    // create a new folder
    // the folder is created in the user's directory at the specified path
    // within the FileInfo object
    // the file information is automatically added to the database and the username
    // is extracted from the token
    @PostMapping("/newfolder")
    public ResponseEntity<String> createFolder(Principal principal, @RequestBody FolderInfo folder) {
        userService.lock(principal.getName());
        String directory = folder.getDirectory();
        if (directory == null || directory.equals("null") || directory.equals("") || directory.equals("\\")) {
            directory = "/";
        }
        String foldername = folder.getName();
        if (foldername == null) {
            userService.unlock(principal.getName());
            return ResponseEntity.badRequest().body("No folder name provided");
        }
        Path path = Paths.get(System.getProperty("user.dir"), "filedir", principal.getName(), directory, foldername);
        // folder already exists
        if (Files.exists(path)) {
            userService.unlock(principal.getName());
            return ResponseEntity.badRequest().body("Folder already exists");
        }
        // create the folder
        try {
            Files.createDirectory(path);
        } catch (IOException e) {
            // replace all back slashes with forward slashes
            String newPath = path.toString().replace("\\", "/");
            path = Paths.get(newPath);
            try {
                Files.createDirectory(path);
            } catch (IOException e2) {
```

```
            userService.unlock(principal.getName());
            return ResponseEntity.badRequest().body("Failed to create folder");
        }
    }
    String folderpath = path.toString();
    folder.setPath(folderpath);
    folder.setOwner(principal.getName());
    folder.setDirectory(directory);
    // add the folder to the database
    folderservice.createFolder(folder);
    userService.unlock(principal.getName());
    return ResponseEntity.ok("Folder was created successfully");
}

// get all folders in the user's directory
// uses the provided token to get the username
// list of files is returned in the 'files' array of the FolderInfo object
@GetMapping("/folders")
public ResponseEntity<List<FolderInfo>> getHomeDirectoryContents(Principal principal) {
    userService.lock(principal.getName());
    String owner = principal.getName();
    List<FolderInfo> home = folderservice.getHomeDirectoryContents(owner);
    userService.unlock(principal.getName());
    return ResponseEntity.ok(home);
}

// list all files and folders of a user
// using data from the database
@GetMapping("/home")
public ResponseEntity<FolderInfo> getHomeDirectory(Principal principal) {
    userService.lock(principal.getName());
    String owner = principal.getName();
    FolderInfo home = folderservice.getFolder(owner, "root", owner);
    if (home == null) {
        userService.unlock(principal.getName());
        return ResponseEntity.badRequest().body(home);
    }
    userService.unlock(principal.getName());
    return ResponseEntity.ok(home);
}

// list all files and folders of a user wrapped using the DirectoryWrapper class
@GetMapping("/homeWrapped")
public ResponseEntity<DirectoryWrapper> getHomeDirectoryWrapped(Principal principal) {
    userService.lock(principal.getName());
    String owner = principal.getName();
    FolderInfo home = folderservice.getFolderByPath(owner);
    if (home == null) {
        userService.unlock(principal.getName());
        return ResponseEntity.badRequest().body(null);
    }
    DirectoryWrapper homeWrapped = new DirectoryWrapper(home);
    userService.unlock(principal.getName());
    return ResponseEntity.ok(homeWrapped);
}

// list all files and folders of a user
@GetMapping("/listFiles")
public ResponseEntity<FileNode> listFiles(Principal principal) {
    userService.lock(principal.getName());
    FileNode root;
    try {
        Path path = Paths.get(System.getProperty("user.dir"), "filedir", principal.getName());
        root = new FileNode(path.toFile());
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Failed to list files", e);
    }
    userService.unlock(principal.getName());
    return ResponseEntity.ok(root);
}
```

```java
// get all files from a folder
// if the directory is null, the user's home directory is used
// uses the provided token to get the username
@GetMapping("/folderFiles")
public ResponseEntity<List<FileInfo>> getFolderContents(Principal principal, @RequestParam String directory) {
    userService.lock(principal.getName());
    if (directory == null || directory.equals("") || directory.equals("/")) {
        List<FileInfo> files = fileService.getAllFiles(principal.getName());
        userService.unlock(principal.getName());
        return ResponseEntity.ok(files);
    }
    String owner = principal.getName();
    List<FileInfo> files = fileService.getDirectoryContents(directory, owner);
    userService.unlock(principal.getName());
    return ResponseEntity.ok(files);
}

// uses the provided token to get the username
@GetMapping("/file")
public ResponseEntity<FileInfo> getFileInfo(Principal principal, @RequestParam String directory,
        @RequestParam String filename) {
    userService.lock(principal.getName());
    if (directory == null) {
        directory = "";
    }
    String owner = principal.getName();
    FileInfo f = fileService.getFile(owner, directory, filename);
    if (f == null) {
        userService.unlock(principal.getName());
        return ResponseEntity.badRequest().body(null);
    }
    userService.unlock(principal.getName());
    return ResponseEntity.ok(f);
}

// uses the provided token to get the username
// requires the directory and foldername of the folder
@GetMapping("/folder")
public ResponseEntity<FolderInfo> getFolderInfo(Principal principal, @RequestParam String directory,
        @RequestParam String foldername) {
    userService.lock(principal.getName());
    if (directory == null) {
        directory = "";
    }
    String owner = principal.getName();
    FolderInfo f = folderservice.getFolder(owner, directory, foldername);
    if (f == null) {
        userService.unlock(principal.getName());
        return ResponseEntity.badRequest().body(null);
    }
    userService.unlock(principal.getName());
    return ResponseEntity.ok(f);
}

// delete a file from the server
// uses the provided token to get the username
@DeleteMapping("/deleteFile")
public ResponseEntity<String> deleteFile(Principal principal, @RequestBody FileInfo file) {
    userService.lock(principal.getName());
    String owner = principal.getName();
    String directory = file.getDirectory();
    if (directory == null || directory.equals("\\\\") || directory.equals("\\\\\\") || directory.equals("\\")) {
        directory = "";
    }
    String filename = file.getName();
    Path path = Paths.get(System.getProperty("user.dir"), "filedir", owner, directory, filename);
    // check if the file exists
    if (Files.exists(path)) {
        // delete the file
```

```
            try {
                Files.delete(path);
            } catch (IOException e) {
                // replace all back slashes with forward slashes
                String newPath = path.toString().replace("\\", "/");
                path = Paths.get(newPath);
                try {
                    Files.delete(path);
                } catch (IOException e2) {
                    userService.unlock(principal.getName());
                    return ResponseEntity.badRequest().body("Failed to delete file");
                }
            }
            // delete the file from the database
            String pathString = path.toString();
            String[] pathArray = pathString.split(owner);
            if (pathArray.length == 1) {
                pathString = owner;
            } else {
                pathString = owner + pathArray[1];
            }
            fileService.deleteFile(pathString);
            userService.unlock(principal.getName());
            return ResponseEntity.ok("File was deleted successfully");
        } else {
            userService.unlock(principal.getName());
            return ResponseEntity.badRequest().body("File does not exist");
        }
    }

    // delete a folder and all of its contents
    // uses the provided token to get the username
    @DeleteMapping("/deleteFolder")
    public ResponseEntity<String> deleteFolder(Principal principal, @RequestBody FolderInfo folder) {
        userService.lock(principal.getName());
        String owner = principal.getName();
        String directory = folder.getDirectory();
        if (directory == null || directory.equals("\\\\") || directory.equals("\\\\\\") || directory.equals("\\")) {
            directory = "";
        }
        String foldername = folder.getName();
        Path path = Paths.get(System.getProperty("user.dir"), "filedir", owner, directory, foldername);
        // check if the folder exists
        if (Files.exists(path)) {
            // delete the folder
            try {
                Files.walkFileTree(path, new SimpleFileVisitor<Path>() {
                    @Override
                    public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) throws IOException {
                        Files.delete(file);
                        userService.unlock(principal.getName());
                        return FileVisitResult.CONTINUE;
                    }

                    @Override
                    public FileVisitResult postVisitDirectory(Path dir, IOException exc) throws IOException {
                        Files.delete(dir);
                        userService.unlock(principal.getName());
                        return FileVisitResult.CONTINUE;
                    }
                });
            } catch (IOException e) {
                userService.unlock(principal.getName());
                return ResponseEntity.badRequest().body("Failed to delete folder");
            }
            // delete the folder from the database
            String pathString = path.toString();
            String[] pathArray = pathString.split(owner);
            if (pathArray.length == 1) {
                pathString = owner;
```

```
            } else {
                pathString = owner + pathArray[1];
            }
            folderservice.deleteFolder(pathString);
            userService.unlock(principal.getName());
            return ResponseEntity.ok("Folder was deleted successfully");
        } else {
            userService.unlock(principal.getName());
            return ResponseEntity.badRequest().body("Folder does not exist");
        }
    }

    // rename a file
    // uses the provided token to get the username
    @PutMapping("/renameFile")
    public ResponseEntity<String> renameFile(Principal principal, @RequestBody FileInfo file,
            @RequestParam String newFilename) {
        userService.lock(principal.getName());
        String owner = principal.getName();
        String directory = file.getDirectory();
        if (directory == null || directory.equals("\\\\") || directory.equals("\\\\\\") || directory.equals("\\")) {
            directory = "";
        }
        String filename = file.getName();
        Path path = Paths.get(System.getProperty("user.dir"), "filedir", owner, directory, filename);
        // check if the file exists
        if (Files.exists(path)) {
            // rename the file
            try {
                Files.move(path, path.resolveSibling(newFilename));
            } catch (IOException e) {
                // replace all back slashes with forward slashes
                String newPath = path.toString().replace("\\", "/");
                path = Paths.get(newPath);
                try {
                    Files.move(path, path.resolveSibling(newFilename));
                } catch (IOException e2) {
                    userService.unlock(principal.getName());
                    return ResponseEntity.badRequest().body("Failed to rename file");
                }
            }
            // rename the file in the database
            String pathString = path.toString();
            String[] pathArray = pathString.split(owner);
            if (pathArray.length == 1) {
                pathString = owner;
            } else {
                pathString = owner + pathArray[1];
            }
            fileService.renameFile(pathString, newFilename);
            userService.unlock(principal.getName());
            return ResponseEntity.ok("File was renamed successfully");
        } else {
            userService.unlock(principal.getName());
            return ResponseEntity.badRequest().body("File does not exist");
        }
    }

    // rename a folder
    // uses the provided token to get the username
    @PutMapping("/renameFolder")
    public ResponseEntity<String> renameFolder(Principal principal, @RequestBody FolderInfo folder,
            @RequestParam String newFoldername) {
        userService.lock(principal.getName());
        String owner = principal.getName();
        String directory = folder.getDirectory();
        if (directory == null || directory.equals("\\\\") || directory.equals("\\\\\\") || directory.equals("\\")) {
            directory = "";
        }
        String foldername = folder.getName();
```

```
        Path path = Paths.get(System.getProperty("user.dir"), "filedir", owner, directory, foldername);
        // check if the folder exists
        if (Files.exists(path)) {
            // rename the folder
            try {
                Files.move(path, path.resolveSibling(newFoldername));
            } catch (IOException e) {
                // replace all back slashes with forward slashes
                String newPath = path.toString().replace("\\", "/");
                path = Paths.get(newPath);
                try {
                    Files.move(path, path.resolveSibling(newFoldername));
                } catch (IOException e2) {
                    userService.unlock(principal.getName());
                    return ResponseEntity.badRequest().body("Failed to rename folder");
                }
            }
            // rename the folder in the database
            String pathString = path.toString();
            String[] pathArray = pathString.split(owner);
            if (pathArray.length == 1) {
                pathString = owner;
            } else {
                pathString = owner + pathArray[1];
            }
            folderservice.renameFolder(pathString, newFoldername);
            userService.unlock(principal.getName());
            return ResponseEntity.ok("Folder was renamed successfully");
        } else {
            userService.unlock(principal.getName());
            return ResponseEntity.badRequest().body("Folder does not exist");
        }
    }

    // user not found exception
    public class UserNotFoundException extends Exception {
        public UserNotFoundException(String errorMessage) {
            super(errorMessage);
        }
    }

    @ExceptionHandler(UserNotFoundException.class)
    public ResponseEntity<String> handleUserNotFoundException(UserNotFoundException e) {
        return ResponseEntity.badRequest().body(e.getMessage());
    }
}
```

PageController.java

```
package proj.skybin.controller;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.servlet.view.RedirectView;

@Controller
public class PageController {

    @GetMapping("/account")
    public RedirectView account() {
        return new RedirectView("/");
    }

    @GetMapping("/about")
    public RedirectView about() {
        return new RedirectView("/");
    }
```

```
    @GetMapping("/login")
    public RedirectView login() {
        return new RedirectView("/");
    }

    @GetMapping("/directory")
    public RedirectView directory() {
        return new RedirectView("/");
    }

}
```

UserController.java

```
package proj.skybin.controller;

import java.security.Principal;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.io.IOException;
import java.nio.file.FileVisitResult;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.SimpleFileVisitor;
import java.nio.file.attribute.BasicFileAttributes;

import proj.skybin.model.AuthRequest;
import proj.skybin.model.UserInfo;
import proj.skybin.service.JwtService;
import proj.skybin.service.UserService;

@RestController
@RequestMapping("/api")
public class UserController {

    @Autowired
    private JwtService jwtService;

    @Autowired
    private UserService userService;

    @Autowired
    private AuthenticationManager authenticationManager;

    // create a new user
    // this will create a new directory for the user
    // this will also create a new account for the user
    // the user will be able to log in after this
    // the home directory will be empty with the directory name being the user's
    // username
    // within the database the home directory's directory will be 'root'
    @PostMapping("/create")
    public ResponseEntity<String> createUser(@RequestBody UserInfo user) {
        UserInfo u = userService.createUser(user);
        if (u == null) {
```

```
            return ResponseEntity.badRequest().body("User already exists");
        }
        return ResponseEntity.ok("User created");
    }

    // authenticate a user and return a token
    // tokens are valid for two hours
    @PostMapping("/authenticate")
    public ResponseEntity<String> authenticateAndGetToken(@RequestBody AuthRequest authRequest) {
        try {
            Authentication authentication = authenticationManager.authenticate(
                    new UsernamePasswordAuthenticationToken(authRequest.getUsername(), authRequest.getPassword()));
            if (authentication.isAuthenticated()) {
                return ResponseEntity.ok(jwtService.generateToken(authRequest.getUsername()));
            } else {
                return ResponseEntity.badRequest().body("Authentication failed");
            }
        } catch (Exception e) {
            return ResponseEntity.badRequest().body("Incorrect username or password");
        }
    }

    // refresh a token
    // tokens are valid for two hours
    @GetMapping("/refresh")
    public String refreshToken(Principal principal) {
        return jwtService.generateToken(principal.getName());
    }

    // delete a user
    // this will delete all files and folders associated with the user
    // this will also delete the user's account
    // this will also invalidate the user's token
    @DeleteMapping("/delete")
    public ResponseEntity<String> deleteUser(Principal principal) {
        userService.lock(principal.getName());
        if (userService.deleteUser(principal.getName())) {
            // delete user's directory
            Path path = Paths.get(System.getProperty("user.dir"), "filedir", principal.getName());
            System.out.println("path: " + path);
            // check if the folder exists
            if (Files.exists(path)) {
                // delete the folder
                try {
                    Files.walkFileTree(path, new SimpleFileVisitor<Path>() {
                        @Override
                        public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) throws IOException {
                            Files.delete(file);
                            userService.unlock(principal.getName());
                            return FileVisitResult.CONTINUE;
                        }

                        @Override
                        public FileVisitResult postVisitDirectory(Path dir, IOException exc) throws IOException {
                            Files.delete(dir);
                            userService.unlock(principal.getName());
                            return FileVisitResult.CONTINUE;
                        }
                    });
                    userService.unlock(principal.getName());
                    return ResponseEntity.ok("user deleted");
                } catch (IOException e) {
                    userService.unlock(principal.getName());
                    return ResponseEntity.badRequest().body("Failed to delete user");
                }
            }
        }
        userService.unlock(principal.getName());
        return ResponseEntity.badRequest().body("User not found");
    }
```

```
    // update username
    // this will update the user's username
    // this will also update the user's directory name
    @PutMapping("/update/username")
    public ResponseEntity<String> updateUsername(Principal principal, @RequestBody String newUsername) {
        userService.lock(principal.getName());
        if (userService.updateUsername(principal.getName(), newUsername)) {
            // rename user's directory
            Path path = Paths.get(System.getProperty("user.dir"), "filedir", principal.getName());
            Path newPath = Paths.get(System.getProperty("user.dir"), "filedir", newUsername);
            // check if the folder exists
            if (Files.exists(path)) {
                // rename the folder
                try {
                    Files.move(path, newPath);
                    userService.unlock(principal.getName());
                    return ResponseEntity.ok("username updated");
                } catch (IOException e) {
                    userService.unlock(principal.getName());
                    return ResponseEntity.badRequest().body("Failed to update username");
                }
            }
        }
        userService.unlock(principal.getName());
        return ResponseEntity.badRequest().body("User not found");
    }
}
```

File Repository Handlers

FileRepository.java

```
package proj.skybin.repository;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.stereotype.Repository;
import org.springframework.data.jpa.repository.JpaRepository;
import proj.skybin.model.FileInfo;
import java.util.Optional;
import java.util.List;



@EnableJpaRepositories
@Repository
public interface FileRepository extends JpaRepository<FileInfo, String>{
    Optional<FileInfo> findByPath(String filepath);
    List<FileInfo> findByOwner(String owner);
    List<FileInfo> findByDirectory(String directory);
    List<FileInfo> findByDirectoryAndOwner(String directory, String owner);
    Optional<FileInfo> findByOwnerAndDirectoryAndName(String owner, String directory, String name);
    void deleteByPath(String filepath);
}
```

FolderRepository.java

```
package proj.skybin.repository;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.stereotype.Repository;
import org.springframework.data.jpa.repository.JpaRepository;
import proj.skybin.model.FolderInfo;
import java.util.Optional;
import java.util.List;
```

```java
@EnableJpaRepositories
@Repository
public interface FolderRepository extends JpaRepository<FolderInfo, String>{
    Optional<FolderInfo> findByPath(String folderpath);
    Optional<FolderInfo> findByName(String name);
    List<FolderInfo> findByOwner(String owner);
    Optional<FolderInfo> findByDirectory(String directory);
    Optional<FolderInfo>findByOwnerAndDirectoryAndName(String owner, String directory, String name);
    List<FolderInfo> findByPathAndOwner(String folderpath, String owner);
    List<FolderInfo> findByDirectoryAndOwner(String directory, String owner);
}
```

SharedFilesRepository.java

```java
package proj.skybin.repository;

import java.util.List;

import java.util.Optional;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.stereotype.Repository;

import proj.skybin.model.SharedFiles;

@EnableJpaRepositories
@Repository
public interface SharedFilesRepository extends JpaRepository<SharedFiles, String> {
    List<SharedFiles> findBySharedUser(String sharedUser);
    Optional<SharedFiles> findByPathAndOwnerAndSharedUser(String path, String owner, String sharedUser);
}
```

UserRepository.java

```java
package proj.skybin.repository;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.stereotype.Repository;
import org.springframework.data.jpa.repository.JpaRepository;
import proj.skybin.model.UserInfo;
import java.util.Optional;


@EnableJpaRepositories
@Repository
public interface UserRepository extends JpaRepository<UserInfo, Long>{
    Optional<UserInfo> findByUsernameAndPassword(String username, String password);
    Optional<UserInfo> findByUsername(String username);
    void deleteByUsername(String username);
}
```

Data Models and SQL interfaces

AuthRequest.java

```java
package proj.skybin.model;

import lombok.AllArgsConstructor;
import lombok.Builder;
```

```java
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class AuthRequest {
    private String username;
    private String password;
}
```

DirectoryWrapper.java

```java
package proj.skybin.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.util.Date;
import java.util.ArrayList;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class DirectoryWrapper {
    private String path;
    private String name;
    private String owner;
    private String directory;
    private Boolean isDirectory;
    private Date uploadDate;
    private ArrayList<DirectoryWrapper> contents;
    private ArrayList<String> sharedUsers;

    public DirectoryWrapper(FolderInfo folderInfo) {
        this.path = folderInfo.getPath();
        this.name = folderInfo.getName();
        this.owner = folderInfo.getOwner();
        this.directory = folderInfo.getDirectory();
        this.isDirectory = folderInfo.getIsDirectory();
        this.uploadDate = folderInfo.getUploadDate();
        this.contents = new ArrayList<>();
        this.sharedUsers = new ArrayList<>();
        if (folderInfo.getSubfolders() != null) {
            for (FolderInfo subfolder : folderInfo.getSubfolders()) {
                this.contents.add(new DirectoryWrapper(subfolder));
            }
        }
        if (folderInfo.getFiles() != null) {
            for (FileInfo file : folderInfo.getFiles()) {
                this.contents.add(new DirectoryWrapper(file));
            }
        }
    }

    public DirectoryWrapper(FileInfo fileInfo) {
        this.path = fileInfo.getPath();
        this.name = fileInfo.getName();
        this.owner = fileInfo.getOwner();
        this.directory = fileInfo.getDirectory();
        this.isDirectory = fileInfo.getIsDirectory();
        this.uploadDate = fileInfo.getUploadDate();
        this.contents = null;
        this.sharedUsers = new ArrayList<>();
        if (fileInfo.getSharedUsers() != null) {
            for (String user : fileInfo.getSharedUsers()) {
```

```
                this.sharedUsers.add(user);
            }
        }
    }
}
```

FileInfo.java

```
package proj.skybin.model;

import java.util.ArrayList;
import java.util.Date;

import com.fasterxml.jackson.annotation.JsonBackReference;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.PrePersist;
import jakarta.persistence.Table;
import jakarta.persistence.Temporal;
import jakarta.persistence.TemporalType;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "files")
public class FileInfo {
    private String name;
    @Id
    @Column(name = "filepath")
    private String path;

    @ManyToOne
    @JoinColumn(name = "folderpath", referencedColumnName = "folderpath")
    @JsonBackReference
    private FolderInfo parent;

    private String type = "file";
    private String parentpath;
    private Boolean isDirectory = false;
    private String owner;
    private String directory;
    private ArrayList<String> sharedUsers;

    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "upload_date")
    private Date uploadDate;

    @PrePersist
    protected void onCreate() {
        this.uploadDate = new Date();
    }
}
```

FileNode.java

```
package proj.skybin.model;
```

```java
import java.io.File;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;

/**
 * This class is for helping index a user's directory recursivly
 */
public class FileNode {
        public final String name;
        public final String path;
        private List<FileNode> contents = new ArrayList<>();
        public final boolean isDirectory;

        private static String getRelativePath(File baseDir, File file) {
            Path basePath = baseDir.toPath();
            Path filePath = file.toPath();

            // Use the relativize method to get the relative path
            Path relativePath = basePath.relativize(filePath);

            // Convert the relative path to a string
            return relativePath.toString();
        }

        /**
         * @param name name of the directory
         * @param contents List of items in the directory
         */
        public FileNode(File root) {
            this.name = root.getName();
            this.path = getRelativePath(Paths.get(System.getProperty("user.dir"), "filedir").toFile(), root);
            this.isDirectory = root.isDirectory();
            if (isDirectory){
                File[] files = root.listFiles();
                if (files != null)
                    for (File file : files){
                        this.contents.add(new FileNode(file));
                    }
            }
        }

        /**
         * returns the name of the FileNode
         * @return
         */
        public String getName() {
            return name;
        }

        /**
         * returns the list of inner files of the folder
         * @return list of FileNodes
         */
        public List<FileNode> getContents() {
            return contents;
        }

        /**
         * adds a list of FileNodes
         * @param contents list of filenodes
         * @return list of filenodes
         */
        public List<FileNode> addContents(List<FileNode> contents) {
            this.contents.addAll(contents);
            return this.contents;
        }

        /**
```

```
         *
         * @param contents FileNode to add
         * @return list of filenodes
         */
        public List<FileNode> addContents(FileNode contents) {
            this.contents.add(contents);
            return this.contents;
        }
    }
```

FolderInfo.java

```
package proj.skybin.model;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import com.fasterxml.jackson.annotation.JsonBackReference;
import com.fasterxml.jackson.annotation.JsonManagedReference;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "folders")
public class FolderInfo {
    private String name;
    @Id
    @Column(name = "folderpath")
    private String path;
    private String owner;
    private String directory;
    private String type = "folder";
    private Boolean isDirectory = true;

    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "upload_date")
    private Date uploadDate;

    @ManyToOne
    @JoinColumn(name = "parentpath")
    @JsonBackReference
    private FolderInfo parent;

    @OneToMany(mappedBy = "parent", cascade = CascadeType.ALL)
    @JsonManagedReference
    private List<FolderInfo> subfolders = new ArrayList<>();

    @OneToMany(mappedBy = "parent", cascade = CascadeType.ALL)
    @JsonManagedReference
    private List<FileInfo> files = new ArrayList<>();

    @PrePersist
    protected void onCreate() {
        this.uploadDate = new Date();
    }
}
```

SharedFiles.java

```java
package proj.skybin.model;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.IdClass;
import jakarta.persistence.Table;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
@IdClass(SharedFilesId.class)
@Entity
@Table(name = "shared")
public class SharedFiles {
    @Id
    private String path;
    @Id
    private String owner;
    @Id
    private String sharedUser;

    private String name;
}
```

SharedFilesId.java

```java
package proj.skybin.model;

import java.io.Serializable;

import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class SharedFilesId implements Serializable {
    private String path;
    private String owner;
    private String sharedUser;

    // equals() and hashCode() methods

}
```

UserInfo.java

```java
package proj.skybin.model;

import java.util.Date;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.PrePersist;
import jakarta.persistence.Table;
import jakarta.persistence.Temporal;
import jakarta.persistence.TemporalType;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "users")
public class UserInfo {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    private String username;
    private String password;
    private String roles = "ROLE_USER";
    @Column(name = "lock_column")
    private Boolean lock = false;

    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "created_at")
    private Date createdAt;

    @PrePersist
    protected void onCreate() {
        this.createdAt = new Date();
    }
}
```

Authentication Filter

JwtAuthFilter.java

```java
package proj.skybin.filter;

import java.io.IOException;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import proj.skybin.config.UserInfoUserDetailsService;
import proj.skybin.service.JwtService;

@Component
public class JwtAuthFilter extends OncePerRequestFilter {

    @Autowired
    private JwtService jwtService;

    @Autowired
    private UserInfoUserDetailsService userDetailsService;

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws
ServletException, IOException {
        String authHeader = request.getHeader("Authorization");
        String token = null;
        String username = null;
        if (authHeader != null && authHeader.startsWith("Bearer ")) {
            token = authHeader.substring(7);
            username = jwtService.extractUsername(token);
```

```
        }

        if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails = userDetailsService.loadUserByUsername(username);
            if (jwtService.validateToken(token, userDetails)) {
                UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(userDetails, null,
userDetails.getAuthorities());
                authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(authToken);
            }
        }
        filterChain.doFilter(request, response);
    }
}
```

Configurations

SecurityConfig.java

```
package proj.skybin.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import proj.skybin.filter.JwtAuthFilter;

@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class SecurityConfig {

    @Autowired
    private JwtAuthFilter authFilter;

    // user details service for authentication
    @Bean
    public UserDetailsService userDetailsService() {
        return new UserInfoUserDetailsService();
    }

    // security configuration
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        // disable csrf to allow post requests (need to find fix)
        return http.csrf(csrf -> csrf.disable())
                // allow requests to the specified endpoints
                .authorizeHttpRequests(authorize -> authorize
                .requestMatchers("/api/create", "/api/authenticate", "/static/**", "/**.html", "/**.js", "/**.css", "/",
"/**.png", "/account", "/**", "/directory", "/about").permitAll())
                // require authentication for all other requests
                .authorizeHttpRequests(authorize -> authorize
                .requestMatchers("/api/**").authenticated())
                // stateless session
```

```
                .sessionManagement(management -> management
                        .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                // specify the authentication provider and filter
                .authenticationProvider(authenticationProvider())
                // add the jwt filter before the username/password filter
                .addFilterBefore(authFilter, UsernamePasswordAuthenticationFilter.class)
                // build the security filter chain
                .build();
    }

    // password encoder for authentication (bcrypt)
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    // authentication provider for authentication
    // uses the user details service and password encoder
    @Bean
    public AuthenticationProvider authenticationProvider(){
        DaoAuthenticationProvider authenticationProvider=new DaoAuthenticationProvider();
        authenticationProvider.setUserDetailsService(userDetailsService());
        authenticationProvider.setPasswordEncoder(passwordEncoder());
        return authenticationProvider;
    }

    // authentication manager for authentication
    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception {
        return config.getAuthenticationManager();
    }

}
```

ServerProperties.java

```
package proj.skybin.config;

import org.springframework.boot.context.properties.ConfigurationProperties;

@ConfigurationProperties(prefix = "server",
    ignoreUnknownFields = true)
public class ServerProperties {
    public static class Tomcat {
        public static class Threads {
            // max # of threads
            private int max = 200;

            public int getMax() {
                return max;
            }
        }
    }
}
```

UserInfoUserDetails.java

```
package proj.skybin.config;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import proj.skybin.model.UserInfo;

import java.util.Arrays;
import java.util.Collection;
import java.util.List;
```

```java
import java.util.stream.Collectors;

public class UserInfoUserDetails implements UserDetails {

    private String name;
    private String password;
    private List<GrantedAuthority> authorities;

    // constructor
    // takes a UserInfo object and converts it to a UserDetails object
    public UserInfoUserDetails(UserInfo userInfo) {
        name=userInfo.getUsername();
        password=userInfo.getPassword();
        authorities= Arrays.stream(userInfo.getRoles().split(","))
                .map(SimpleGrantedAuthority::new)
                .collect(Collectors.toList());
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return authorities;
    }

    @Override
    public String getPassword() {
        return password;
    }

    @Override
    public String getUsername() {
        return name;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

UserInfoUserDetailsService.java

```java
package proj.skybin.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Component;

import proj.skybin.model.UserInfo;
import proj.skybin.repository.UserRepository;
```

```java
import java.util.Optional;

@Component
public class UserInfoUserDetailsService implements UserDetailsService {

    @Autowired
    private UserRepository repository;

    // load a user by username for authentication manager
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        Optional<UserInfo> userInfo = repository.findByUsername(username);
        return userInfo.map(UserInfoUserDetails::new)
                .orElseThrow(() -> new UsernameNotFoundException("user not found " + username));

    }
}
```