Introduction

This app provides various services for managing files and folder on a remote shared server. It is built using a combination of server-side Java Spring controllers and client-side technologies such as HTML, CSS, JavaScript, and jQuery. Users can browse, rename, download, upload, and delete files and folder on the remote server using a simple and intuitive user interface. The app was designed with performance in mind and implements asynchronous Promise-based communication with the server. Additionally, it is deployed using the Apache Tomcat web server, which ensures that it can be accessed by the user through a web browser on their computer.

To operate the app on your computer, you will first need to get Apache Tomcat installed. Once you get Tomcat installed and configured, you can start the server by running the appropriate command or script for your operating system. Once the server is running, open your web browser and navigate to the URL for the app, typically http://localhost:8080/. This should lead you to the app's homepage, where you can start using its features to browse, rename, download, upload, and delete files and sub-folders on a remote shared folder. You can use the navigation buttons to move between functionalities. For more information on how to use the app, a readme.txt is provided upon starting the app.

Development Approach

In designing this app, I chose to concentrate on the server-side implementation first before working on the client-side. This approach allowed me to create the basis for the app's functionality, ensuring that it could handle a broad range of file management tasks on the remote server with ease. By using server-side Java Spring controllers, I produced a backend system that could communicate with the remote server and execute tasks such as file browsing, renaming, downloading, uploading, and deletion. Once the server-side functionality was fully tested, I turned my attention to designing the client-side interface using HTML, CSS, JavaScript, and jQuery. This approach ensured that the client-side was built around a strong foundation, making it easy to produce a seamless and intuitive user experience that leveraged the the server-side implementation.

Requirements:

We would like to take control of a shared folder under a remote system. More specifically, we would like to be able to:

- Browse the remote shared folder, including its sub-folders, asynchronously.
- Rename a remote shared file or sub-folder, asynchronously.
- Download a remote shared file, asynchronously.
- Upload a local file to a specified path under the remote shared folder, asynchronously.
- Delete a shared remote file, asynchronously.

The objective is to design and develop a suitable solution using a Pseudo RESTful Java/Spring server on the remote system and a JavaScript Promise-based asynchronous consumer.

Additionally, the app needs to provide an intuitive and user-friendly interface that could be accessed through a web browser on the user's computer.

Server-Side Development:

Design: I designed the Java Spring provider to offer the required service for managing files and folders on a remote shared server. These services include browsing, renaming, downloading, uploading, and deleting files and folders.

Implementation: To implement the Java Spring provider, I used a variety of technologies including Apache Tomcat web server, Java Spring framework, and Spring modules such as Spring Boot. I faced various challenges during the implementation, including issues with connecting to the remote server and handling large file uploads and downloads.

To overcome these challenges, I used debugging tools such as logging and breakpoints to identify and repair errors in the code. I also implemented optimizations such as chunked file transfer and progress tracking to improve the performance of the app when handling large file uploads and downloads.

Client-Side Development:

User Interface: The website layout was designed to be simple and easy to navigate, with a clean and modern design aesthetic. I used JavaScript with JQuery plugin, Featherlight, to do so. I was able to create a website that provided an intuitive and user-friendly interface for accessing the various file management services provided by the Java Spring provider.

Integrating the server-side provider with the client-side website:

To integrate the server-side Java Spring provider with the client-side website, I used a Pseudo RESTful API architecture. I exposed the Java Spring provider's functionalities as Pseudo RESTful services that the client-side site could consume using JavaScript's Fetch API. The services were implemented using Spring's @RestController annotation, which allowed me to define the endpoints and map them to the Java Spring provider's services.

During integration, I faced various challenges such as cross-domain resource sharing and handling the response formats from the server-side provider. To overcome these challenges, I added CORS (Cross-Origin Resource Sharing) to the server-side provider's responses to provide cross-domain resource sharing between the client-side site and the server-side provider.

Technology Enablers

I leveraged the power of the Spring Framework to develop the backend of the web application. Spring provides a robust and scalable platform for developing enterprise-level applications using Java. I created Spring controllers to provide Pseudo RESTful services to the web application.

These controllers receive requests from the frontend and return responses in the form of JSON data.

JavaScript and Promises to handle asynchronous operations in the web application. Promises provide a clean and intuitive way to manage the flow of asynchronous code and avoid callback hell. They provide a programming abstraction used to represent the eventual completion (or failure) of an asynchronous operation and its resulting value. I used them in the JavaScript code to handle the asynchronous requests to the Spring controllers and crucially to consume the Spring controllers asynchronously.

Apache Tomcat: I used the Apache Tomcat web server to deploy and run the Spring application. Tomcat is a lightweight and highly configurable web server that provides a robust and scalable platform for Java web applications.

NetBeans IDE to develop the Java-based web application and Visual Studio Code to write JavaScript and other frontend code.

HTML to structure the content of the web application and CSS to style its content. I used jQuery as a JavaScript library and Featherlight as a jQuery plugin.

Conclusion and Known Bugs

In conclusion, I have designed and developed a full-stack Spring web application that provides services for browsing, renaming, downloading, uploading, and deleting remote shared files and sub-folders. The design involved both server-side development using Java Spring and client-side development using HTML, CSS, JavaScript, and JQuery. I use a sequential approach to development, starting with the server-side provider and then integrating it with the client-side website. The server-side development approach involved identifying the project requirements, designing the Java Spring provider's architecture, functionality, and features, and implementing it using tools such as Apache Tomcat and NetBeans IDE. The client-side development approach involved identifying the website requirements, designing the website's layout, navigation, and user interface, and implementing it using tools such as Visual Studio Code. Integration was achieved using a Pseudo RESTful API architecture, CORS, and the HTTP protocol. The integration process required careful planning, design, and testing to ensure that the client-side site could seamlessly communicate with the server-side Java Spring provider.

Known Bugs:

1.Renaming a file inside a sub-folder fails.
2.Attempting to navigate to a sub-folder immediately after renaming it fails, but reloading the page does fix the issue.

Credits

The design of this website was inspired from: https://criticalmediartstudio.com/witnessing/ by criticalmediartstudio.