

**Homework 3: Technology enablers**

**Design document**

**by**: Fatima El kabir

**supervised by**: Dr. Omar Iraqi

**Table of content:**

1. **Objective**
2. **Development Approach**
3. **User manual**
4. **Objective**

The primary objective of this assignment is to empower the user with the capability to assume command over a remote server. Within this context, the user is expected to have the ability to perform a range of essential actions on the remote shared server. These actions encompass tasks such as browsing the server's contents, renaming files, downloading files, uploading files, and deleting files as necessary. In essence, the assignment aims to provide the user with comprehensive control and management over the remote server's files and directories, thereby facilitating a seamless and efficient interaction with the server's resources.

1. **Development approach**

**Server-Side Functionality Overview**

In the server-side component, we've leveraged the same provider utilized in the previous deliverable. We've fine-tuned this provider to enhance its capabilities, particularly in the areas of renaming files and efficiently browsing subfolders. To provide a comprehensive understanding, here's a detailed breakdown of the various functions within the provided code and the methods used in their development.

1. **Browse\_Files(String path):** This function serves the purpose of listing all files within a specified directory. It utilizes the File class to interact with the file system, extracting vital file information. The method takes an input path, constructs the corresponding directory path, and verifies the existence of the path as well as its nature as a directory. Once these conditions are met, it compiles a list of files and generates a FileInfo object for each file. FileInfo, in this context, is a class that serializes file-related information such as name, size, and type. These FileInfo objects are then added to an ArrayList, and, in the end, the function returns an array of FileInfo objects as the response.
2. **Rename\_File(String path, String newname):** This function is designed to rename a file or directory based on the specified old path and the new name. It checks for the existence of the file or directory and whether the new name already exists in the parent directory. If these conditions are satisfied, the file or directory undergoes renaming via the renameTo() method. The function returns an appropriate message reflecting the success of the operation.
3. **Download\_File( String filename, String path)**This function facilitates client-side downloads of files from the server. It starts by confirming the existence of the file and employs a BufferedInputStream to read the file's content into a ByteArrayResource. The function also configures the necessary HTTP headers for the download and delivers the ByteArrayResource as the response.
4. **Upload\_File (MultipartFile file, String path):** This function empowers clients to upload files to the server. It carefully validates the file's non-emptiness, the specified path's status as a valid directory, and the absence of a file with the same name. When these conditions align, an InputStream and FileOutputStream are employed to copy the content of the uploaded file into the server's file system. The function, in turn, communicates a message indicating the outcome of the operation.
5. **Delete\_File (String path):** This function grants clients the ability to delete files from the server. It rigorously checks for the file's existence and ensures it's not a directory. When these conditions are met, the file undergoes deletion through the delete() method. The function, similar to the others, communicates a message reflecting the success of the operation.

These functions are mapped to appropriate HTTP request methods (GET, POST, DELETE) and are associated with specific URL paths through the use of Spring Web annotations. The adoption of ResponseEntity objects guarantees the correct HTTP status codes and response bodies are consistently delivered to address various scenarios effectively.

**Consumer-Side Interaction:**

On the consumer side, users engage with the server-side functionalities through an intuitive command-line interface. This client code is crafted in JavaScript and leverages the generated client stub for making RPC (Remote Procedure Call) requests to the server. The various functions within this client code operate in tandem with the server in the following ways:

1. **browse()**
   * **Purpose:** Allows the user to browse a directory on a server.
   * **Implementation:**
     + Creates a promise that resolves when the browsing is complete.
     + Uses readline to get user input for the folder path.
     + Invokes the corresponding RPC method on the client stub to browse the directory on the server.
     + Processes the server response and logs file information.
     + Resolves the promise.
2. **rename()**
   * **Purpose:** Renames a file or directory on the server.
   * **Implementation:**
     + Creates a promise that resolves when the renaming is complete.
     + Uses readline to get user input for the old and new names.
     + Invokes the corresponding RPC method on the client stub to rename the file or directory on the server.
     + Resolves the promise based on the success or failure of the operation.
3. **download()**
   * **Purpose:** Downloads a file from the server to a local directory.
   * **Implementation:**
     + Creates a promise that resolves when the download is complete.
     + Uses readline to get user input for the file name and path.
     + Invokes the corresponding RPC method on the client stub to initiate the file download from the server.
     + Retrieves the file from the server, determines the download path, and streams the file to the local system.
     + Resolves or rejects the promise based on the success or failure of the download.
4. **upload()**
   * **Purpose:** Uploads a file to the server.
   * **Implementation:**
     + Creates a promise that resolves when the upload is complete.
     + Uses readline to get user input for the file name.
     + Reads the file content using fs.readFile.
     + Invokes the corresponding RPC method on the client stub to upload the file to the server.
     + Resolves or rejects the promise based on the success or failure of the upload.
5. **deleteFile()**
   * **Purpose:** Deletes a file or directory on the server.
   * **Implementation:**
     + Creates a promise that resolves when the deletion is complete.
     + Uses readline to get user input for the file or directory name.
     + Invokes the corresponding RPC method on the client stub to delete the file or directory on the server.
     + Resolves or rejects the promise based on the success or failure of the deletion.
6. **menu()**
   * **Purpose:** Displays a menu for the user to choose an operation.
   * **Implementation:**
     + Outputs a menu to the console.
     + Uses readline to get user input for the chosen option.
     + Executes the corresponding function based on the user's choice.
     + Continues the menu until the user chooses to quit.

**Main Execution:** Calls the menu() function to start the interactive menu. In summary, the consumer-side functionalities harmoniously interact with the server, offering users a seamless and efficient experience in managing files and directories. The use of promises in each function helps manage asynchronous operations and handle success or failure in a structured way. The recursive calls to the menu functions after each operation ensure the user can continue interacting with the program until they choose to quit.

1. **User manual**

To start, initiate the server-side code by following these steps:

1. Navigate to the "src" directory by executing: **cd path/to/src**.
2. Compile the project using the command: **gradle build**.
3. Launch the Spring Boot server with: **gradle bootrun**.

Once the server-side operation is complete, proceed with the client-side tasks:

1. Go to the "JavaScript" directory by running: **cd path/to/code**.
2. Execute the command: **node Consumer.js** .
3. Then the menu will be prompted, and the user can chose the functionality he/she likes:

A screenshot of a computer

Description automatically generated