**Design Document**

**Purpose of the Application**

The application allows you to take control of a shared folder under a remote system. The client can browse folders and subfolders, Delete, download, rename and upload files. The client is able to call functions as if they were local thanks to RPC paradigm.

**Running the Java Provider**

1- Create a new terminal and go to the project newdemo.

2- Build the project using: ./gradlew build

3- Run the project using: ./gradlew run

**Running the Python Consumer**

1- Create a new terminal and go to the project newdemo.

2- Type python app/src/main/Python/Client.py to reach the client.py file and run it

The application will show the following menu:

Available Services:

1- Browse a file

2- Rename a file

3- Delete a file

4- Download a file

5- Upload a file

6- Quit

- The application will prompt the user to enter a choice

**The Application Service**

The client can browse folders and subfolders, Delete, download, rename and upload files:

* Browse Folders and sub-folders
* Delete a remote shared file
* Download a remote shared file
* Rename a remote shared file
* Upload a file under the remote shared folder

**API**

The API is the contract between the service provider and the service consumer.

The methods that the client can invoke are:

- Browse(DirectoryName)

The method takes as an argument the directory name that the user wants to browse.

To go back to the parent folder the user enters “./”

If the user wants to go back to the main folder (ServerShare), the user dosen’t have to enter anything.

The method return a list containing the name of files and subfolders contained in the folder the user wants to browse

If no file is found an empty list is sent

- Rename(PreviousName, NewName)

The method gets as arguments the name of the file the user wants to rename, and its new name.

The method returns true if the rename operation is successful or false otherwise.

In order for the operation to succeed the user should be browsing in the folder where the file exists.

-Download (Filename)

The user enter the name of the file he wants to download

The methods return an array of bytes containing the bytes of the file

If the file is not found the method raises an exeption.

In case of any problem the method return an empty array.

In order for the operation to succeed the user should be browsing in the folder where the file exists.

- Upload (byte[], path)

The method takes as argument the bytes of the file the user wants to upload and the path where the user wants to upload the file in the remote shared folder

If the operation is successful the method returns true, otherwise it returns false.

If the file is not found an exception is raised

-Delete(filename)

The method takes as an argument the filename the user wants to delete

If the operation is successful the method return true, else it return false.

In order for the operation to succeed the user should be browsing in the folder where the file exists.

**Development Process (Java First)**

Since I am not able to write the API using WSDL, I adopted the Code-first approach using Java. Then using the wsgen tool, I was able to generate the API in WSDL.

For the client side, I used the Zeep module in order to generate the client stub, then the menu that the user will use to access the services.

**Code Structure**

Service provider:

This method consist of the business implementation of the methods that user will call as if they were local. The provider side consist of two main files:

- FileBrowser.java: Contains the real implementation of the methods and annotated with @WebService annotation.

-Provider.java: Publishes the Web Service. The Skeleton is responsible for establishing connection with client side, marshalling/Unmarshalling of data.

Service consumer:

In this part the user will be able to call function as if they were local. the client stub class is responsible for establishing connection with client side, marshalling/Unmarshalling of data.