جامعة الأخوين

ⵜⴰⵙⴷⴰⵡⵉⵜ ⵏ ⵓⵅⴰⵡⴰⵢⵏ

AL AKHAWAYN UNIVERSITY

GDrive Homework #2

Design Document

CSC 3374

Dr. Omar Houssaini Iraqi

Submitted in

27th of March 2022

by

Youssef Gaimes

1. Purpose of the Application:

This client-server application allows taking control of a server directory in order to back-up files and synchronize that directory across many clients. The system allows:

- Uploading files to the server
- Receiving files from the server
- Performing CRUD operations on files in the client side

2. Technology Enablers:

Since the project relies on the client-server architecture, I decided to develop my solution using the REST architecture in order to be able to leverage HTTP verbs and map them to custom functions. This approach also allows for easy debugging functionalities such as breakpoints and line by line execution. In compliance with the homework's requirements, I have used a Spring based application as a server side as well as python as a client side. My python application leverages mostly the requests library to send HTTP requests to the server side.

3. End-User Manual:

3.1. Running the server:

The server is a Springboot Gradle application using the REST architecture to map to the verbs provided by the HTTP protocol. In order for the server to function correctly, we would need to specify the path to our desired folder to act as the server-side folder and then running the Gradle application with the following cli commands:

- gradle build
- gradle bootRun

3.2. Running the client:

The client also needs for the path to be changed to an existing directory to act as the consumer directory. In order to run the python client, please run the following cli command:

- Python consumer.py

4. Protocol:

1. The server initializes and listens to the default port 8080.
2. The client starts by synchronizing its directory with the server by sending an HTTP GET request to the server using the Requests library. The server, upon reception of the request, send back a JSON hash table of all the files existing in the specified server directory and an MD5 hash value of the specified file with the following structure:

   {

   “file name” : ”file hash”

   }
3. The client initializes by checking if the folder it is pointing to is empty, if it is, we would assume that the client is a new client, and it downloads all the files from the server using a GET request to /file and providing the filename as a parameter for each key in the hash table.
4. The client then checks if its directory has the same files with the same checksum every 10 seconds. If not, it sends a PUT request mapped to /file with the file name as

a parameter to the server specifying the file it would like to upload. The server then deletes the already existing file if it exists and rewrites it.

5. The client also checks if the hash table sent by the server is the same as files in the client folder. If not, it sends a DELETE request to the server with the file name as a parameter.

5. GDrive Structure:

The server side runs first and starts accepting requests through the port 8080. It is imperative for the good functioning of the application to specify a folder to use using its absolute path.

Depending on what the client would like to do, it sends one of three verbs which are:

a. GET verb:
    i. "/" mapping:

       the following route prompts the server to execute the sync function and return a hash table of the files names existing in the server directory as a key and their md5 hash as a value.

    ii. "/file" mapping:

       This route requires a file name parameter. It returns the array of bytes of the specified file.

b. PUT verb:
    i. "/" mapping:

       The following route requires a file name and its array of bytes. The server deletes the file if it already exists, creates a new one and writes the byte array into it.

c. DELETE verb:

    i. “/” mapping:

        This route requires a file name. It searches for it in the server directory and deletes it.