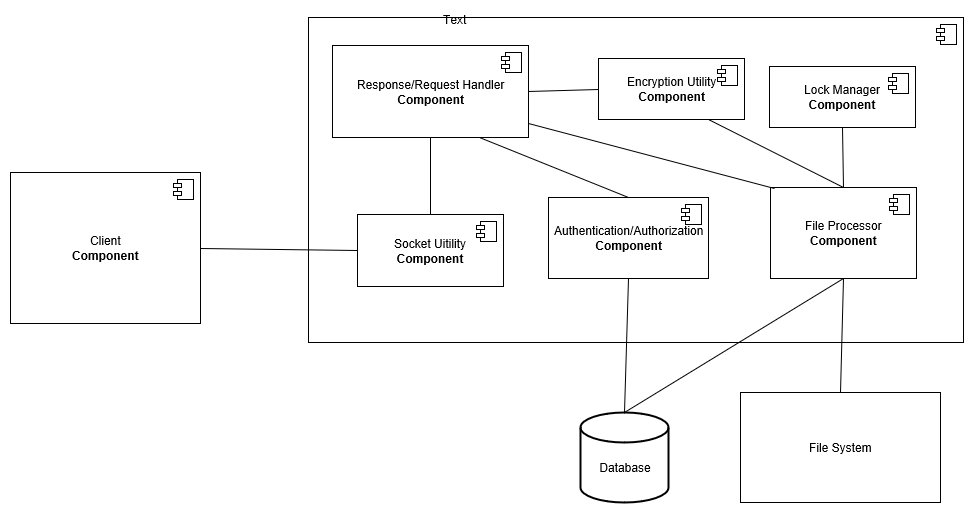
**Introduction:**

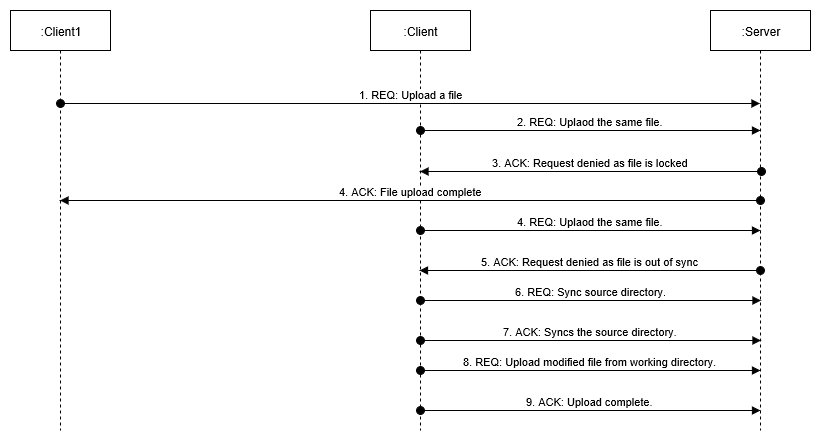
A client-server application which would enable user to sync files and folders across his/her multiple devices. The user will run client application and login to application. User can start uploading files now. As soon as the file from one device is uploaded to server, the same file is delivered to users all devices.

**Architecture:**

**Component diagram:**



**Sequence diagram:**



**Workflow of the application:**

1. Server is started
2. User installs client application on all the devices that he/she wishes to be synced.
3. User provides few arguments to start the application (username, password, source directory to be tracked)
4. If the login is successful, application starts
5. As soon as the application starts, the files from the server is synced to local device
6. After start-up, user has 2 options:
   1. Upload file
   2. Exit – shutsdown the client application
7. To upload file, system will prompt for filename that is to be uploaded. This path will be relative to source directory
8. The file or folder provided by user is validated against various checks
   1. If file/folder exists
   2. If extension is valid
   3. If size is allowed
9. If validations are successful, a check is performed to check if file is actually different from file present at server. This check helps to reduce reuploading if same version of file is present on server
10. On above successful validations, the file is sent to the server. If it’s a folder, the files present in folder is recursively checked and sent to server
11. At the same time, if some other device tries to upload the file, the file is locked and the second uploader is informed appropriately
12. As the file is being uploaded, the old version still remains accessible, as the old file wont be overwritten until the new version is completely transferred. This also helps to avoid data corruption, loss of data and to tackle against network failures.
13. Once the file is completely received by server, the older version is deleted and overwritten
14. After the transfer, the same file is shared with all the devices of the user signed in. This is done parallelly to make file available to all devices as soon as possible
15. Once transfer is completed, file is unlocked to be uploaded by other devices.

**Technical Implementations**

User Registration/ Login :

The user registration and login is not validated in the implementation of this application as the main focus is on implementation was on implementation of syncing capabilities

Communication:

Socket communication is used to communicate between clients and server. This help to transfer data at faster rate since the transfer is transport layer protocol. Since data is gonna be huge, socket communication is better over other alternatives like web services.

File transfer:

When the user selects the file to upload, the files size and type is validated. To block any type of file being uploaded, it can be configured in the FileConfig.properties file.

Whenever the file is being uploaded, to avoid transfer of same file again, the checksum is calculated of the file present at the client. This is then compared with the checksum of the file present on server. If both the checksum matches, the file is not transferred. SHA-1 algorithm is used to find the checksum.

Upload of large files:

Upload of large files is not handled in this application because of the strict time-constraint. To handle upload of large files we can suggest 2 strategies:

1. Send the files in small chunks. This will enable to avoid retransferring of files in case if network fails
2. Zip the file before sending and unzip it after receiving. This will help to reduce the load on servers and network. Java provides inbuilt APIs to zip and unzip the files.

Concurrency control on parallel access of file:

When one file is being uploaded by one device, other device wont be able to upload the same file until the first uploader has finished uploading.

This concurrency is maintained by implementing a locking mechanism. When a file is being uploaded, the file is locked and locked is released after the file is delivered to all servers

In this application, lock is maintained by putting an entry of filename and user in a hashmap. If an entry is present in hashmap, file is considered to be locked. The hashmap is managed by LockManager class which is made Singleton, so that only one entity is maintaining the lock. Bill-Pugh Singleton implementation is used.

Parallel delivery of files to multiple devices:

When one user uploads a file, same file is to be updated in all devices. As soon as file is received by server, the same file is sent to all other devices of user parallelly. This is implemented using different threads assigned to each client. Each thread in the thread pool delivers file to each device.

Controlled access of files on server:

The file on the server is stored in a file system manner. The control of access to files can be done using checking for rights on the folder. Each users data is stored in different folder in the root folder. This helps to segregate different users data and controlled access to the files.

Handling multiple requests to the server at same time:

When server receives multiple requests from different clients, each request thread is handled by different processor thread. This leads to handle multiple requests at the same time, hence getting optimum use of the underlying hardware.

**Configurable Parameters**

Different parameters that can be configured in the application are

ApplicationConfig.properties

NO\_OF\_FILE\_RETRY\_ATTEMPT=3

SERVER\_IP=localhost

SERVER\_PORT=8034

SOURCE\_DIRECTORY=C://Users//Sandesh//Desktop//MangoAppsDir//Client//de

LISTEN\_PORT=8001

DEVICE\_ID=device1

FileConfig.properties

FILES\_NOT\_ALLOWED=exe

MAX\_FILE\_SIZE\_ALLOWED=10000000

ClientDetails.properties

admin=127.0.0.1:8001:device1;127.0.0.1:8002:device2;

**Testing procedure**

Both the executable client and server jars are placed in MangoApps/Executable folder. For both, the properties folder is present at the path MangoApps/Executable/Client/properties and MangoApps/Executable/Server/properties respectively

* First of all **start the server**
* Before stating the server, you can configure the PORT on which your server will be listening for requests.
* Configure the SOURCE\_DIRECTORY to a location where you want server to store files of all users.
* Configure the user details that will be connecting to server in ClientDetails.properties

The data is stored in following manner:

User**=**device1\_ip:device1\_port:device1\_name**;**device2\_ip:device2\_port:device2\_name

* You can start the server by executing runServer.bat present in MangoApps/Executable/Server folder

**Starting the clients:**

* You can start as many instance of clients as you want
* Configure the SOURCE\_DIRECTORY to a location where you want client to store files or to upload from.
* Ensure that all clients are listening on different ports:

You can configure it in ApplicationConfig.properties of each Client

LISTEN\_PORT=8001

* Configure DEVICE\_ID in ApplicationConfig.properties. This device id should be same as configured in server’s client details properties

DEVICE\_ID=device1

* Configure SERVER\_IP and PORT to communicate with server

SERVER\_IP=localhost

SERVER\_PORT=8034

* You can start the server by executing runClient.bat present in MangoApps/Executable/Client folder

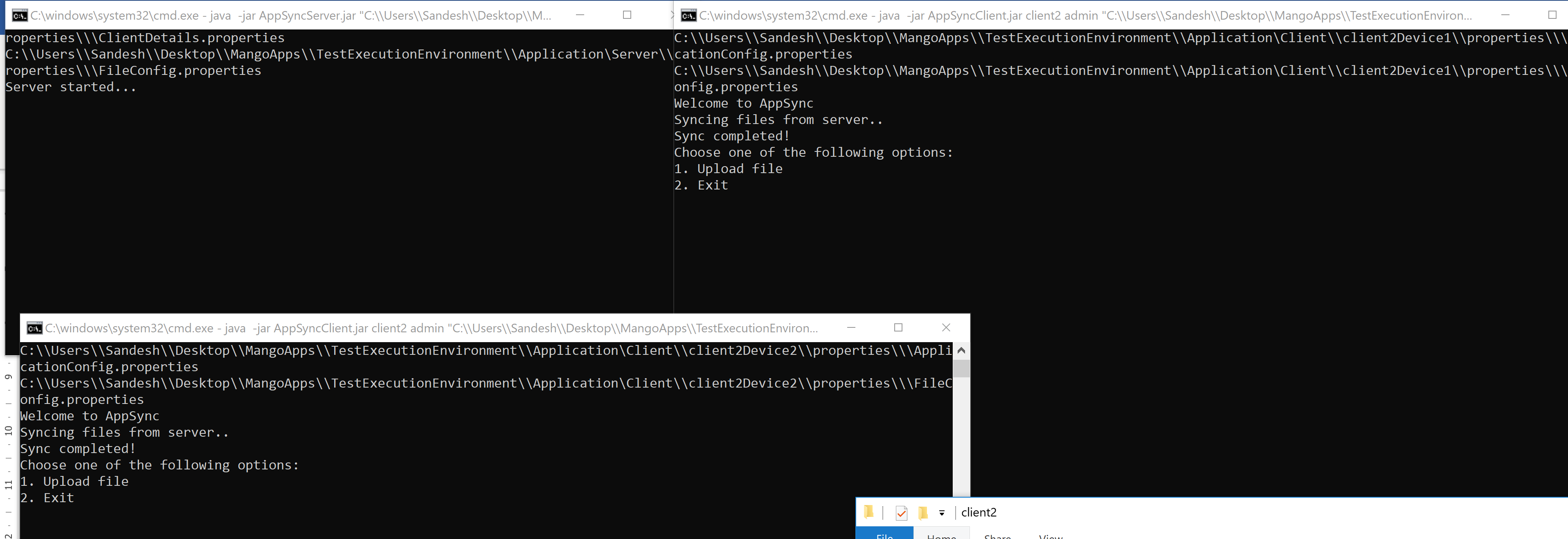
For easy testing, a bat script is provided which will start 1 server and 5 clients. 3 for devices of ‘admin’ user and 2 for ‘client2’ user.

Edit the bat script to provide SOURCE\_DIRECTORIES for client and server

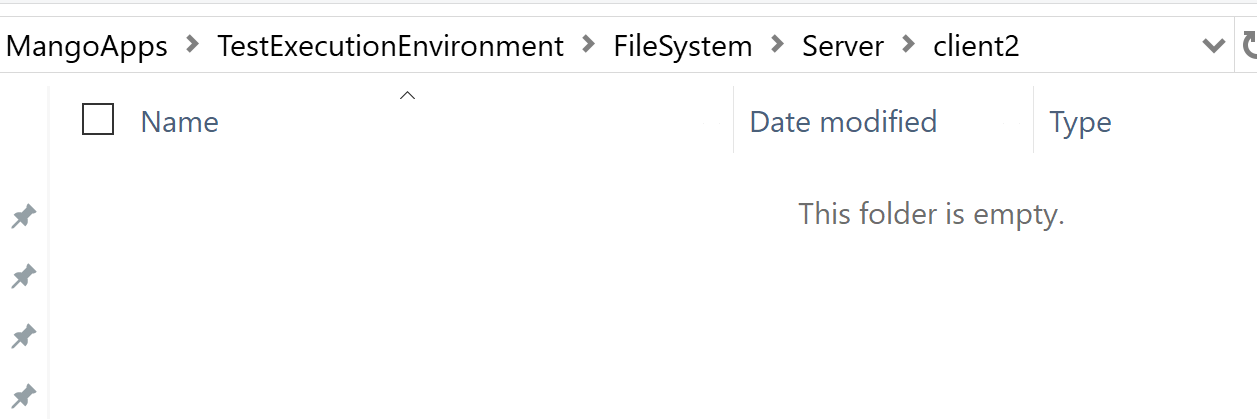
Upload file in any of the device using the relative path from SOURCE\_DIRECTORY and then check if file is present in SOURCE\_DIRECTORY of other devices of same user. Check SOURCE\_DIRECTORY of server as well

**Screenshots**

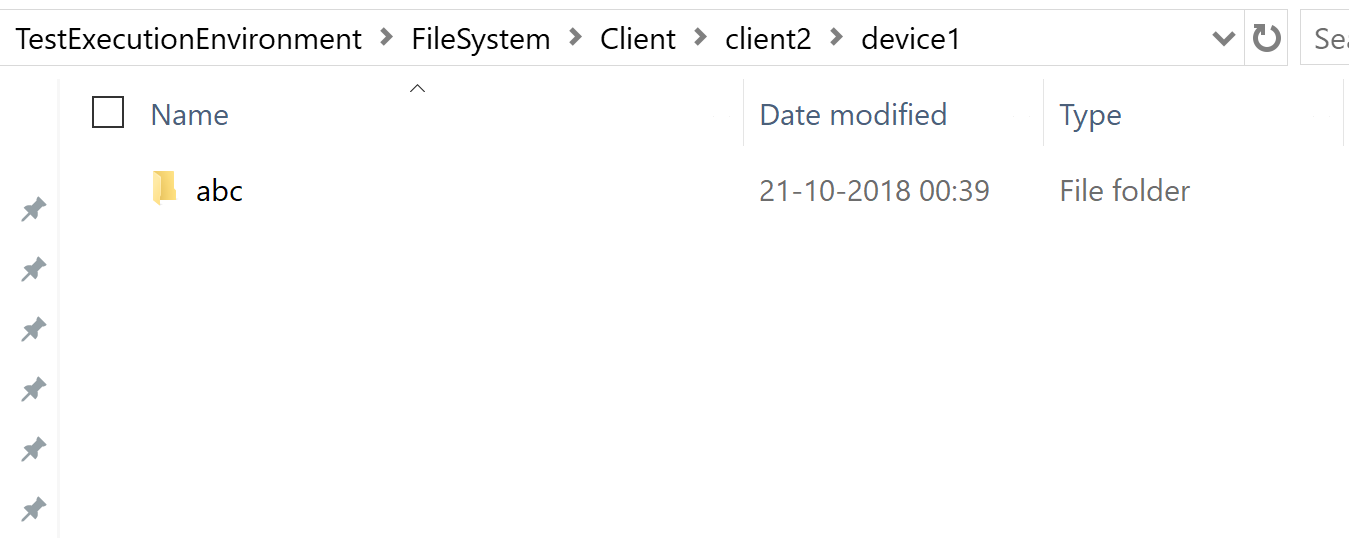
1. Server and 1 client(2 devices) started



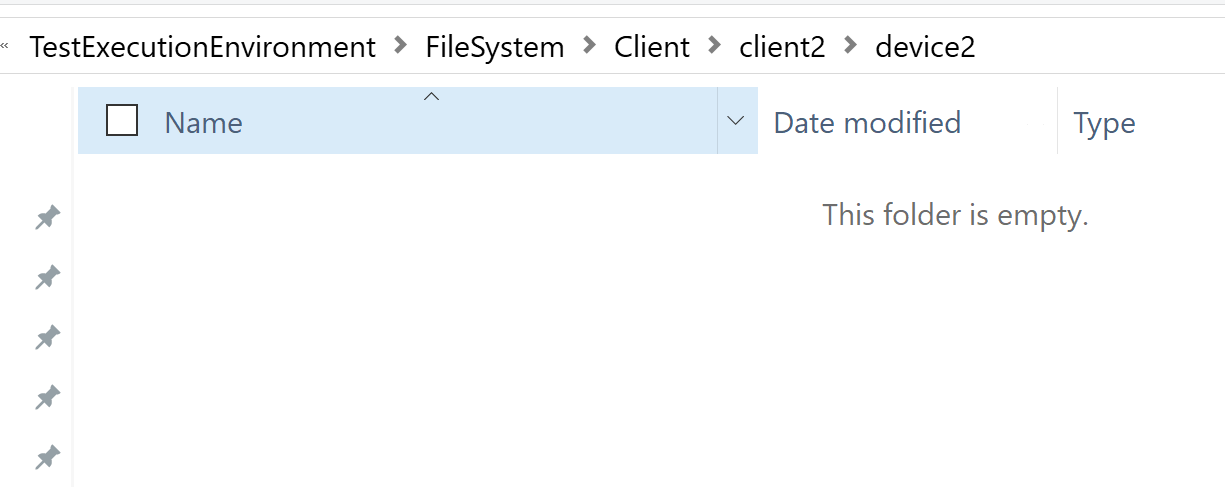
1. Server doesn’t have any data of ‘client2’



1. ‘device1’ of ‘client2’ has folder abc



1. ‘device2’ of ‘client2’ doesn’t have folder abc



1. ‘device1’ uploads folder ‘abc’ to server. Folder gets uploaded to server. Server syncs data with ‘device2’ of ‘client2’

