Group Project - 7CCSMGPR

Deadline Fighters Final report - March 24, 2019

1 Desktop client implementation

We made a planning framework when we realized the functionality of the desktop client. We firstly mapped out implementing upload files, download files, delete files, rename files, edit files and other functions. Secondly, implementing the ability to combine available functions into one file synchronization feature. Finally, we will beautify the interface and improve user experience.

In the process of implementation, the main process is divided into the following steps:

Table 1: Steps of processes

Step	Process
1	Configure the computer to connect to AWS S3.
2	Create a new electron project.
3	Open the project, configure the Access key ID and Secret Access Key in code and make the project connect to S3.
4	Add a file selector and initially implement the function of uploading a single file to S3.
5	Initial download of a file with a specific name.
6	Preliminary deletion of files with specific names.
7	Rename the file on the desktop client and upload it to S3.
8	Try to use E-tag and MD5 to edit the file and incrementally upload the changed part. Then plan to compare the local file and the last modified time of the server file to upload the modified file, but it is not successful.
9	Create a folder called deadlinefighters on the local disk.
10	List all files stored on the server.
11	Add button function to every listed file.
12	Click on each individual file, jump out of the two dialogs, download and delete, and perform related operations for specific files.
13	Plan to simplify syncing, first upload all local files then download all server files.
14	Get a list of local files and use a loop to add an upload function to each file.
15	Get a list of server files and use a loop to add a download method to each file.
16	Add upload all function and download all function to the sync function click event.
17	Add CSS.

At the step 4, upload is the first function in file synchronization system that I tried to achieve, html5 had an input method that type = *file*, it will generate a file selector and users can choose a file that in the local disk. I got the element of file in the scripts and put the element in parameters, the parameters that S3 need usually are file name, file type and file content, and file should be uploaded according to the desired format what S3 provided. S3 has an upload method and when desktop sends the parameters, the server will catch and return the error or upload file successfully.

At the next step, I firstly added a download function and that can download a special file, like *1.txt*, from server to local. I used *fs* module in node.js, the *fs* module is the file system module, which is responsible for reading and writing files, it needs to introduce *fs* module in code before use, like *var fs = require(fs)*. I got the bucket (a storage system in S3) and key (file name) of a file, stored them in a parameter and passed it to *getObject* function in S3 to got the whole element of file. Then I used *fs.writeFileSync(filename, data, [options])*, filename will give a name of file and a path that where this file will be located.

```
//download
var fs = require('fs');
var FilePath1 = '/2.docx';
var bucket1 = 'deadlinefighters';
var key1 = '2.docx';
var download1 = new AWS.S3();
download.addEventListener('click', function(){
    var downloadFile = (FilePath1, bucket1, key1) => {
    var params1 = {
        Bucket: bucket1,
        Key: key1
    };
        download1.getObject(params1, (err, data) => {
        if (err) console.error(err)
        fs.writeFileSync(FilePath1, data.Body)
    })
}
downloadFile(FilePath1, bucket1, key1);
        });
```

Figure 1: download code

At the step 7, because S3 had no direct rename method, I only can copy a special file on the server first and give it a new name, then delete the original file.

As for *edit* function, in our plan, we will handle edited files with delta synchronization. Delta sync is used to synchronize the modified portion of the file instead of re-uploading the entire file, in order to increase the speed of server processing. Delta sync relate to MD5 Algorithm and E-tag.But in the end *edit* did not succeed.

Considering that the downloaded files need to be stored in a folder for users conveniently view, I tried to add a method to automatically create a folder in local. JavaScript is not allowed to access local files, but *fs* module can help to deal with it. I used *fs.mkdir* function to create a primary directory named *deadlinefighters*.

In order to better displayed the files on the server, we designed to list all the files. I cited jQuery to simplify code, firstly, used *listObjects* function which S3 was provided to list all the files in the backstage. Then, got the key (file name) and rendered it to the front-end interface. *$.each(objects, function(i, content)* is used for traversing the array, the array here is objects, which contained all the file contents, content represents each file, content.Key shows the name of each file.

```
//list
  bucket.listObjects({}, function(err, data) {
    if (err) console.log(err, err.stack); // an error occurred
    $('#objectList').empty();   //empty the list
    var objects = data.Contents;
    $.each(objects, function(i, content) {
      console.log(data);  //successful response in console
      //$('<li>').text(content.Key).appendTo($('#objectList'));  //list the file name to the page
      $('#objectList').append("<button  id="+i+" value= '"+content.Key+"'>"+content.Key+"</button><br>");

});
```

Figure 2: download code

I changed each line of file to a button and that was convenient to add click events for download and delete functions later.

I set two dialogs to one file, one was download function, the other was delete function. The download function was similar with the previous code, key and file path need to be change, just passed the file values obtained by clicking the button to them. The file following represented key and got the corresponding file object from dialog, the file path pointed to the newly created folder.

var file = $(this).val();
var filePath = '/deadlinefighters'+'/' + file;

The delete function is something different, when the server file is deleted, the local file will be deleted too. Therefore, I added a loop algorithm to traverse local files for each deleted server file, when the name of the local file is equal to the name of the server file, the local file will also be deleted.

```
    fs.readdir('D:\\deadlinefighters\\',function(err,files){
    $.each(files, function(i, filecontent) {
    fs.readFile('D:\\deadlinefighters\\'+filecontent, function (err, data) {
   if (err) {
       return console.error(err);
   }
        else if(filecontent=file){
        fs.unlink('D:\\deadlinefighters\\'+filecontent, function(err){
        if(err){
        console.error(err);
         }
    console.log('file:'+filecontent+'delete sucessfully');
          });
      }

    });
  });
});
```

Figure 3: download code

Because the files existed under a specific folder, it needed to first specify to the folder directory and then used the loop to get information about each file. At first, when read each files name, I didnt add the *'D:\\deadlinefighters\\'+* before

However, the result returned an error, *filecontent* represented all the data in a file, not only a name, so it cant compare with the file name in server.

The step 13 to 16 mainly completed a simple synchronization function. *Upload all* and *download all* functions were all involving loop. *Upload all* was similar with *delete*, just added an upload method to the file loop. *Download all* was unlike with local file loop, the method used here is to loop through the server's files and add the download method.

In general, the desktop client of the file synchronization system has the following buttons and functions:

Table 2: Buttons and functions

Button	Function
sync	Upload all files from a local folder to server. Download all files from server to local folder.
upload	Upload another one file from local to server.
each line of file	List each line of file in server.
download	Download one file.
delete	Delete one file.