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 beautified the interface and improved 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 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	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.
18	Package the whole client to a desktop application.

At the step 4, upload is the first function in file synchronization system that we tried to achieve, html5 had an input method that type = *file*, it would generate a file selector and users can choose a file that in the local disk. We 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, Letao firstly added a download function and that can download a special file, like *1.txt*, from server to local. Letao 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")*. We 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 Letao 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, we 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, we tried to add a method to automatically create a folder in local. At first, Letao tried to use the *FileSystemobject*, it was an object to process files and folders. But JavaScript always used *ActiveXObject* to create a *FileSystemObject* action file, but *ActiveX* only worked on Internet Explorer. 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*. But we found the path should be set different on Mac and Windows system, Siva used *navigator.platform* to check what kind of system it is and add diverse formats to build path of folder.

In order to better displayed the files on the server, we designed to list all the files. We used *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

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

Letao 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, Letao 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 file's name, Letao didn't 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 can't 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.

2 Evaluation

Comparing the design part to the implementation part, here is a table of what design functions are worked and what are not:

As for the desktop client, the *edit* function was not worked. At the beginning, we planned to use the incremental synchronization to handle the files which were changed contents, it means not uploading the whole files but only uploading the part that are different with the original files. We planned to use MD5 to calculate file's value and transfer the MD5 value

Table 3: Priority of operations

Operation	Object	Whether it is completed
Upload	One file (May not in the local sync folder)	Yes
	All files in the local sync folder	Yes
Download	One file	Yes
	All files	Yes
Delete	One file	Yes
	All files	No
Rename	One file	Yes
	Two or more files	No
Edit	One file	No
	Two or more files	No
Synchronize	All files in the server and local syn folder	Yes

to the server, the server determines whether the value of *MD5* is different from the original file.

At first, Letao tried to use the *MD5* API of *node.js*, but got a problem when detecting the *MD5* value of the local file, because JavaScript as a web language, it has no access to check all the parts of local files.

Then, we tried to use last modified time of files to substitute *MD5*, if the last modified time of local file was different with that on server, the file would be re-uploaded. However, desktop client and mobile client both got some problems with database. For the desktop client, we tried to build a database table of file name and last modified time to store files' information on server, but there was a problem that at the beginning, our desktop client use sync = upload all files on local + download all file on server, therefore, the last modified time always be changed. For the mobile client, the last modified time on server had different format with that on local and had an impact on the comparison of the two sides.

The second part that we not completed was conflicts. At this moment, the Android client only can connect with the S3, not the server application, so the conflicts cannot be solved temporarily. We used Amazon S3 at first, but we found S3 had their own methods to solve conflicts, so we changed to *@be/http-service* to build local server.

There are the processes that we did well. Our steps were from simple to difficult, like first upload only one file then upload plenty of files in one folder at the same time. The organized steps made our project had a clear structure.

The *upload all* function finally can work automatic, that means the files be added to the *deadlinefighters* folder can be automatically uploaded to server.

Our desktop client can run on different computer system, like Mac or Windows. It has a strong adaptability.

3 Test

In the project, our team conducted a black box test on the file synchronization tool. At the beginning, our team tested the transmission of different kinds of file names and file formats. As shown below (Table Black box test result at the beginning). As you can see from the table, we have file names under different language specifications, file names with special symbols, and different types of file formats, such as zip, png, and docx etc. used to do the upload, download, delete and rename function tests. For the uploading, downloading and deleting of files, all the objects we tested can be implemented successfully. However, for the rename function, file names with Chinese names and special symbols cannot be renamed.

At the end of the project, our system implemented automation synchronize function, and these files with Chinese names and special symbols finally achieved in rename function. We analyzed the reason why we couldn't achieve it before because the name of the renamed file needs to be written in the code when we originally developed the product function, and the name with Chinese characters may be inconsistent with the language environment in which the code is written. So, the function cannot be achieved.

Furthermore, we also learned and configuration test environment of Mocha, Chai and Sinon for the desktop application. We try to implement the jsunit test. But for some reasons, it is not complete.

Test case is show below:

No.	File type	File successfully uploaded	File successfully download	File successfully deleted	Action of rename	File successfully renamed
1	Test.txt	T	T	T	Rename to Test.docx	T
2	测试.txt	T	T	T	Rename to Ceshi.txt	F
3	123.txt	T	T	T	Rename to 321.txt	T
4	!.txt	T	T	T	Rename to?.txt	F
5	Test.pdf	T	T	T	Rename to 100000.jpg	T
6	Test.doc	T	T	T	Rename to Test 111111.docx	T
7	Test1！.zip	T	T	T	Rename to 测试 2！.zip	F
8	3.png	T	T	T	Rename to three.png	T
9	Test.rtf	T	T	T	Rename to 1.rtf	T

4 System Overview

4.1 Brief Introduction of System Function

With the increasing popularity of network technology and the importance of information construction, people increasingly need a file synchronisation tool to help them get the file they want on different devices. File synchronisation tools make it possible to edit the same files across multiple computers in a sensible way. This system is a 'hub and spoke' file synchroniser which have a single central server (the 'hub') to which multiple other clients (the 'spokes') synchronise. Developing such a tool can bring a lot of convenience to the users as it can let them get the files they need, anytime, anywhere.

Therefore, our team, *Deadline Fighters*, are aims to develop a multi-host file synchronizer which can:

- Allow upload and download of files from a central server (hub) through client applications with necessary authorization.
- Automatically synchronize changes made by client applications unto the corresponding copy of the file in the server.
- Handle all possible conflict/non-conflict scenarios of file synchronization (ie. combinations of Create, Edit, Rename, Delete) involving multiple client applications.
- Enable file synchronisation through both desktop (all platform) and mobile (Android) clients.

as we set in the initial report.

Through the outline of initial report, we detailed each function. Therefore, the file synchronisation tool that our team aim to develop will achieve the following functions as designed during the process: Upload a file from local which is outside the local sync folder; Upload files from local sync folder; Download a file from server; Download files from server; Delete a file/files from local sync folder; Delete a file/files from server; Rename a file/two or more files from local sync folder; Rename a file/two or more files from server; List all files on the server; Edit a file from server; Edit two or more files from server; Synchronize server and local files.
What's more, the file synchronisation tool can be used in both desktop (all platform) and mobile (Android) clients.

Table 4: Priority of operations

Operation	Object	Priority	Description
Upload	One file (May not in the local sync folder)	1	Priority description:
	All files in the local sync folder	2	1. Functions Necessary and Important
Download	One file	1	2. Characteristic Functions
	All files	2	3. General functions
Delete	One file	2	4. Functions can be supplemented
	All files	3	5. Additional functions
Rename	One file	2	
	Two or more files	3	
Edit	One file	4	
	Two or more files	4	
Synchronize	All files in the server and local syn folder	2	

4.2 System user role

There are two types of clients using file synchronisation tools, desktop client (all platform) and mobile client (Android).

Desktop (all platform) clients:

Table 5: Desktop (all platform) client role

Client name	User role
Desktop (all platform)	View files on the server
	Upload a file from local which is outside the local sync folder
	Upload files from local sync folder
	Download a file/files from server
	Delete a file from local sync folder
	Delete a file from server
	Rename a file from local sync folder
	Rename a file from server
	Edit a file from server
	Edit two or more files from server
	Synchronize server and local files

Table 6: Mobile (Android) client role

Client name	User role
Mobile (Android)	View files on the server
	Upload a file from local which is outside the local sync folder
	Upload files from local sync folder
	Download a file/files from server
	Delete a file from local sync folder
	Delete a file from server
	Rename a file from local sync folder
	Rename a file from server
	Edit a file from server
	Edit two or more files from server
	Synchronize server and local files

4.3 Central server's conflict role

For central server: it will solve the conflict between multiple users operate at same time. The operate and resolution is shown as follow table.

(Add conflict table here.)

By using the conflict role as above in the file synchronisation tool development allows the system to reduce the number of error reports and make the system organized. It won't cause the system to lose files and report error when the user using two different synchronization functions at the same time to the server (eg. A client edit a file on the desktop application and the client delete the file on the mobile at the same time). It can avoid unnecessary data losses. Because the most important thing about a synchronous system is that you can't lose files. Through the table listed above, our team group can have a common standard for dealing with conflicts in the development process, making the development process more organized.

According to the above detailed requirements design and contradictory solutions, it can help our team have a better assignment for the tasks to the team members and promote the teamwork. Team members can arrange their own time according to their own tasks to complete system functions implementation.

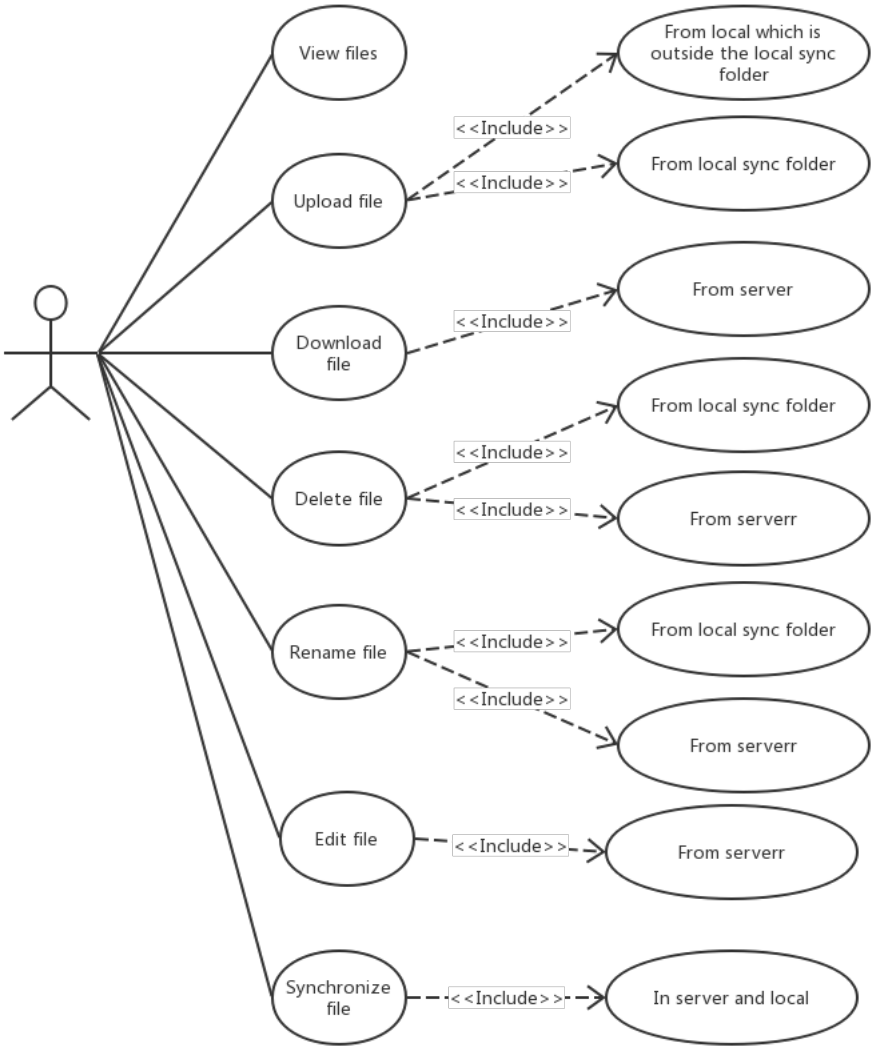
5 User Scenario

5.1 User Overall Scenario Description

Table 7: User Overall Scenario Description

Role	Need	Scenario
Users	View	Files on the server
	Upload	A file from local which is outside the local sync folder
		Files from local sync folder
	Download	A file/files from server
	Delete	A file from local sync folder
		A file from server
	Rename	A file from local sync folder
		A file from server
	Edit	A file from server
		Two or more files from server
	Synchronize	Files in server and local

User use case:



5.2 User Sub-scenario Description

Scenario 1 View files on the server

Scenario 2 Upload a file from local which is outside the local sync folder

References

[1] Amazon S3, [online]Available at: https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/S3.html(Accessed: 3 February 2019).

[2] Upload files on S3, [online]Available at: https://blog.csdn.net/qq1147093833/article/details/80267542(Accessed: 6 February 2019).

[3] MyDatahack(2018),'Uploading and Downloading Files in S3 with Node.js'. [online] Available at: https://www.mydatahack.com/uploading-and-downloading-files-in-s3-with-node-js/(Accessed: 6 February 2019).