File Synchronizer using Python (Server), Electron (Desktop) and Java (Mobile)

By team **Deadline Fighters**

Submitted for 7CCSMGPR - Group Project on March 28, 2019

Georgii Fidarov - 1816977

Letao Wang - 1823375

Lili Chen - 1863966

Qilin Zhou - 1822373

Sivaranjani Subramanian - 1845664

Special thanks to Ka Hang Jacky Au Yeung

1 Aim

Our team, *Deadline Fighters*, set out with the aim 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 sychronization through both desktop (all platform) and mobile (Android) clients.

The desktop application is developed using Electron framework to enable cross-platform compatibility. Based on feedback received during the preliminary presentation, we have developed a Python server with which our client applications interact. Python server, in turn, uses AWS S3 for storage and other purposes.
Of the above mentioned aims, we have been able to accomplish most of it with the following changes/exceptions:

- There is no authorization mechanism present on client applications, except for knowing which IP address to access. Python server, on the other hand, makes requests to AWS S3 after authentication.
- Automatic synchronization has been achieved using FileObserver libraries. It works well for regular use cases and bugs have been logged for other cases.
- Basic dev-tested conflict resolution mechanism implemented using ETag for files as created by AWS S3.

Also, some effort have been put into the aesthetics and usability of the application UI using CSS.

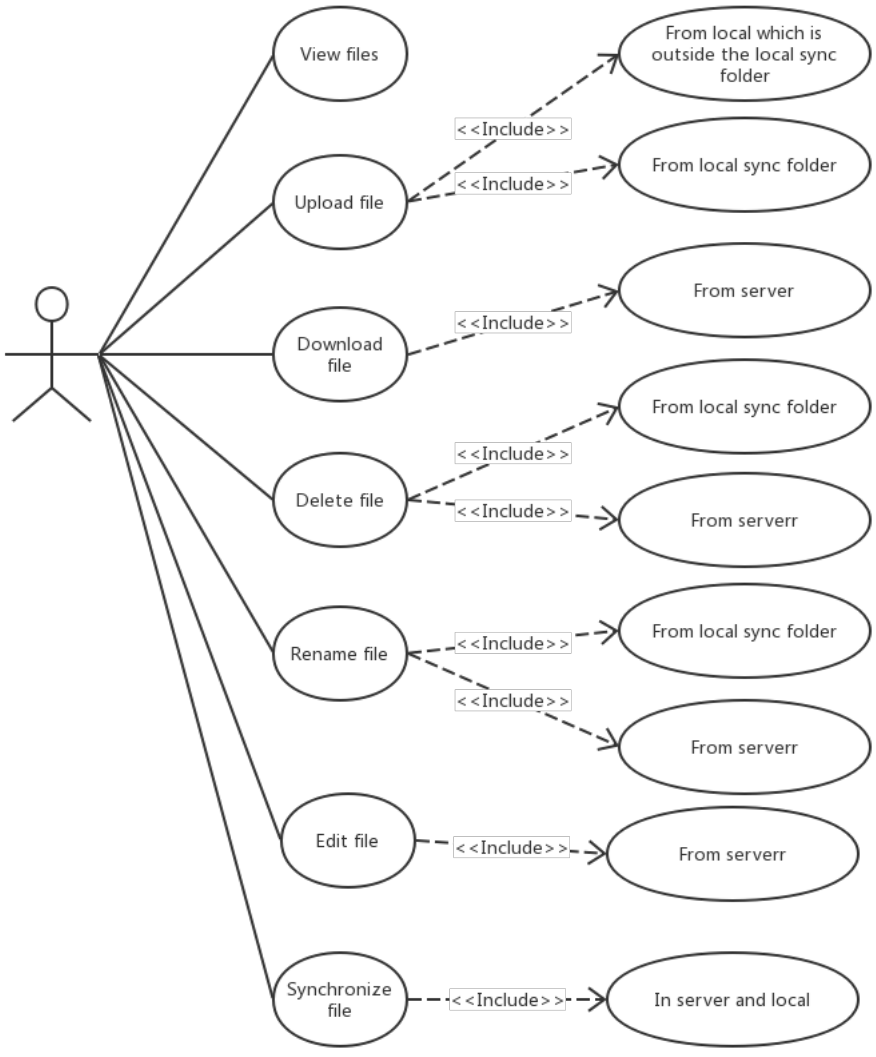

Figure 1: Functionalities provided by Deadline fighter application to Users

The following are links to documentations that we have referenced during this project. All links were accessed during the period January 18, 2019 to March 28, 2019.

References

[1] Amazon S3 SDK for Javascript, *https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/S3.html*

[2] Boto3 for S3 on Python,
https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html

[3] Upload files on S3, *https://blog.csdn.net/qq1147093833/article/details/80267542*

[4] MyDatahack(2018),'Uploading and Downloading Files in S3 with Node.js'
https://www.mydatahack.com/uploading-and-downloading-files-in-s3-with-node-js/

[5] SQLite Python, 'Tutorial for using Sqlite on Python', *http://www.sqlitetutorial.net/sqlite-python/*
Tutorial Tutorial on API calls to AWS S3, *https://grokonez.com/android/uploaddownload-files-images-amazon-s3-android#5_Connect_to_Amazon_S3*

[6] Upload to server using URL for Android, *https://androidexample.com/Upload_File_To_Server_-_Android_Example/index.php?view=article_discription&aid=83*

[7] Change listview height dynamically, *https://stackoverflow.com/a/26501296*

2 System Design

2.1 Introduction

The team *Deadline Fighters* was given the overall requirement to build a *hub and spoke* file synchroniser which has a single central server (the *hub*) to which multiple other clients (the *spokes*) can synchronise. The following are the operations that it entails and the priority of implementation we set for our team.

Client name	User role	Priority	Description
Desktop and Mobile	View files on the server	2	Priority description:
	Upload a file from local which is outside the local sync folder (only for Android)	1	
	Upload a single file from local sync folder	2	**1-Highest priority**
	Upload multiple files from local sync folder	2	
	Download a single file from server	1	(Important /necessary functions)
	Download multiple files from server	2	
	Reflect delete of a single file (Local to server)	2	
	Reflect delete of a single file (Server to local)	2	**4-Lowest priority**
	Reflect delete of multiple files (Local to server)	3	
	Reflect delete of multiple files (Server to local)	3	(Will implement if time permits)
	Reflect rename (Local to server)	2	
	Reflect rename (Server to local)	3	
	Download edited changes on a single file from server	3	
	Download changes on two or more files from server	4	
	Synchronize all server and local files	2	

Table 1: Functionalities to be enabled on client applications

2.2 Architecture

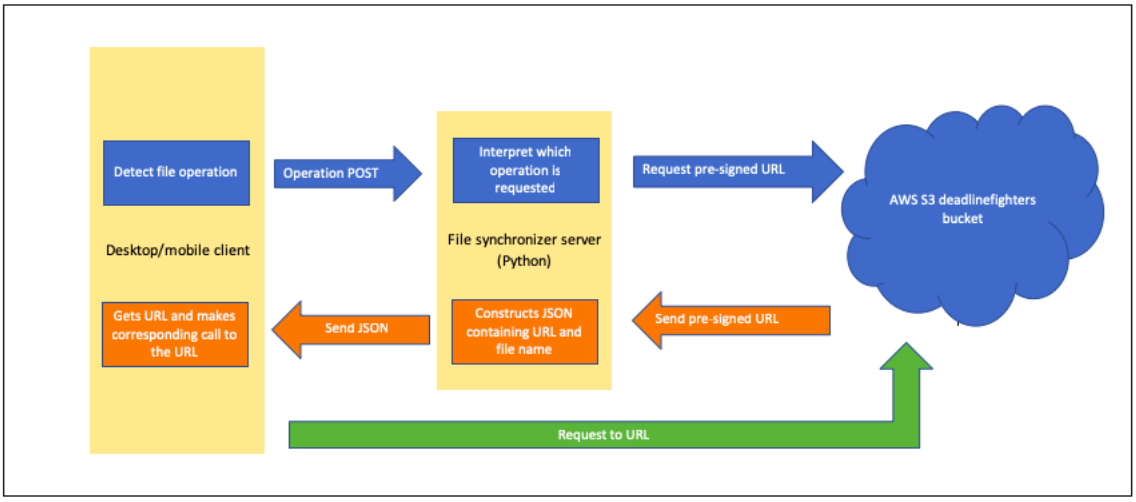

Figure 2: Flow of operation across client, server and AWS S3

Our desktop and mobile clients sends a custom POST request to the DeadlineFighters Python server based on the operation and file involved. The server then interprets the call and makes necessary presigned URL fetch call to our AWS S3 bucket which serves as our cloud storage. The server then passes on the fetch URL to the application. The application now makes necessary HTTP call to the URL based on the operation and payload.

All interactions between the application and the server happens as HTTP requests with JSON request/response body. Using Python server as a proxy to AWS S3 instead of directing interacting with S3 was a decision made after the preliminary presentation. This is in par with industry standards where applications use S3 and other cloud storage but the client applications are unaware of it. This allows for future migration to other cloud endpoint if necessary.

2.3 Conflict resolution

We also leverage the ETag generated by AWS for each file to check the version of the file and detect conflict. We have implemented Sqlite Database on the server which updates its rows with (filename, eTag and lastModified) values upon every upload/download request from client. This helps in detecting potential conflict.

The client will have a Database that has the same schema but the eTag will be the last known eTag for the file. This eTag is sent along with the request to the server. The server compares the eTag sent by the client and the one it has from S3. If they match, it indicates no conflict. If they do not, a conflict window will be popped with options for users to upload local copy, download server copy (both by force) or keep two copies of the file.

At the time of this report submission, development of conflict resolution on desktop client is underway. Conflict resolution will not be implemented on Android during the project timeline.

3 Implementation

The initial team of 6 was divided into sub-teams of 2, each handling desktop client, mobile client and the server component. At this point, server component was AWS S3 and the client applications made direct calls. After team restructuring, development of local server began while the client applications continued to build the system to talk to AWS S3. Desktop, mobile and server code were pushed into corresponding feature branch and then merged into the master branch at regular intervals. This allowed for better testing and narrowing down of issues.

Once the basic decisions about the server were made and functionalities implemented, client applications were migrated to interact through the local server. This delayed progress which we could have used into feature building if we had made better decisions on the server.

Development process was agile with weekly scrum meetings on Thurdays. Emphasis was laid on writing maintainable features and code. Efforts were directed into code reviews, logging for client and server and documenting commonly faced issues. Bugs were tracked under *issues* on our Git repository.

The following are the versions of softwares, packages and IDEs used:

Desktop Application Development

- Framework: Electron 4.0.1
- Dependencies: Chokidar (for file/folder watching), dns, ip, httpreq (for http request)

Mobile Application Development

- IDE: Android Studio 3.3
- Required API: equal to or greater than 25
- Build system: Gradle

Server Development

- Language: Python 2.7
- Database: Sqlite

Cloud endpoint

- Service: Amazon Web Services Simple Storage Service (AWS S3)

4 Testing

For most duration of the project, testing was blackbox and manual. Tests were conducted for synchronization of various file names (characters) and formats. This brought out bugs in json formatting, file name encoding and HTTP request format.

Furthermore, we also learned and configured test environment of Mocha, Chai and Sinon for the desktop application. We tried to implement the jsunit test. Due to lack of time to research more, it could not be completed.

Some of the specific issues found on the desktop client, include:

- PDF file with characters such as space,-," can be downloaded but cannot be opened. Other PDFs can.

- File name with ’ can be deleted in the local folder but cannot be deleted by using the delete button which invokes server delete.
- File with special symbol*& can be uploaded but cannot be downloaded.
- In addition to non-standard named symbols, file names with special symbols (non-ASCII) cannot be renamed, but a new file with those symbols in their name can be uploaded.
- Batch delete/upload does not work.

The test case results are as shown below for Desktop client:

	desktop		
File	upload	download	delete
Test.txt	T	T	T
123.docx	T	T	T
测试.pdf	F	F	F
!,,!.jpg	T	T	T
测试 test.gif	F	F	F
Test!!!.png	T	T	T
123.mp3	T	T	T
2019df.mp4	T	T	T
!,,!.zip	T	F	T
Hello.rar	T	T	T
Test!!!.exe	T	T	T
Test.html	T	T	T
2019df.js	T	T	T
'!,,!' .ppt	T	F	F
Test123.db	T	T	T

(a) Upload,Download,Delete

	Rename to	result	edit
Test.txt	123.txt	T	T
123.docx	测试.docx	F	T
测试.pdf	!,,!.pdf	F	F
!,,!.jpg	测试 test.jpg	F	T
测试 test.gif	Test!!!.gif	F	F
Test!!!.png	123.png	T	T
123.mp3	2019df.mp3	T	T
2019df.mp4	!,,!.mp4	T	T
!,,!.zip	Hello.zip	F	T
Hello.rar	Test!!!.rar	T	T
Test!!!.exe	Test.exe	T	T
Test.html	2019df.html	T	T
2019df.js	'!,,!' .js	T	T
'!,,!' .ppt	Test123.ppt	F	T
Test123.db	Test.db	T	T

(b) Edit and Rename

The test case results for Mobile Client are as shown. Adding files are not as simple on Mobile as it is on Desktop and thus, the test cases are different:

File type	extension	upload	download
text	txt	No error	
document	doc		
	pdf	No error	
	ppt		
image	png	It shows the file in the server, and the according to the size it has upload succeed. But it does not display the actual picture as it should be.	
	gif	Same problem.	
	jpg	Same problem.	
audio	Mp3		
videos	Mp4		

5 Team Contributions

As mentioned, our development process was agile. All team members met for an hour on Thursday 1pm to communicate the progress of each sub-group and the plan for the next week. The main location is on the sixth floor of the Bush house SE building. The meetings were also used to alleviate the difficulties encountered in our development process. The members of the group helped each other to solve these difficulties.

The weekly meeting for Android is scheduled for Friday afternoon, and the desktop meeting is scheduled on Wednesday afternoon. WhatsApp was the chosen medium of informal communication. Independent progress was encouraged so as to not be slowed down by other members. All contributions raised through Pull requests on Git were reviewed by at least two other people to foster peer review and sharing of knowledge.

Below are detailed accounts of the contribution of each member in the team (as written by the team members themselves) during project development process:

(c) Sivaranjani Subramanian

(d) Letao Wang

(e) Qilin Zhou

(f) Lili Chen

Figure 3: Commits and frequency graph of each member

Georgii Fidarov:

- Researched about edit button.
- Researched about AWS Lambda.
- Researched about Flutter.io.
- Demonstrated Conflict scenarios and resolutions during preliminary presentation.

Letao Wang:

- Researched about Amazon S3.
- Researched about using Electron and front-end language(HTML/Javascript) on S3.
- Wrote "Tools/Technology" part in the initial report.
- Demonstrated the basic functions of the desktop client in the initial presentation.
- Implemented the basic function of Deadline Fighters desktop client with AWS S3 as server endpoint: Included upload one file, download a special file, delete a special file, rename a special file(Function removed upon further decisions).
- Researched about delta sync, E-tag, MD5 and database.
- Tried to implement the edit function with delta sync.
- Researched about jQuery.
- Implemented Deadline Fighters desktop client with AWS S3 as server endpoint: Included the download all files, upload all files, delete and download a special file(Function removed upon further decisions) for both server and local storage by using jQuery.
- Designed the UI of the Deadline Fighters desktop client.
- Designed the test case for both the desktop client and mobile client.
- Test the test case for the desktop client.
- Wrote "Implementation of the desktop client" and "Evaluation" part in the final report.

Lili Chen:

- Researched about using the electron and react-native to develop the desktop.
- Wrote 'what we have done so far' and design the desktop UI in the initial report.
- Research about how to using javascript and HTML with AWS S3.
- Implemented Deadline Fighters desktop client with AWS S3 as server endpoint: Included the upload file to server and list file from the server to desktop.
- Added CSS to the desktop.
- Researched database for the desktop login page.
- Researched Rsync algorithm for edit function
- Researched about JQuery, font-awesome icon and bootstrap.
- Researched jsunit test (mocha, sinon, chai) for desktop and try to implement it.
- Done the black box test for the desktop.
- Wrote 'requirement and design', 'test case' and 'team works' in the final report.

Qilin Zhou:

- Write 'team protocol' in the initial report.
- Researched about using Flutter.io.
- Researched the rsync algorithm.
- Researched the unit test of Android and try to use it (not implemented yet).
- Researched and compared different APIs provided by AWS SDK.
- Researched the thread in Android development.
- Implemented Deadline Fighters mobile client with AWS S3 as server endpoint: Included the download, upload, delete and rename(Function removed upon further decisions) for both server and local storage.
- Worked with the desktop client team about the file listing and listener.
- Researched about FileObserver and use FileObserver to detect the file status in the local storage.
- Designed the UI of the Deadline Fighters mobile client.
- Implemented Deadline Fighters mobile client with python server endpoint: Included the download, upload(Functions should be tested and improve).
- Write the 'mobile implementation' in the final report.

Sivaranjani Subramanian:

- Single-handedly coded the Python server and migrated the entire Desktop client and its functionalities to communicate with the server as an enhancement over Letao and Lili's work that used S3 as server. Work included implementing boto3 SDK on Python for talking to AWS, understanding various ways of communication between server and client and implementing the three component framework (AWS, Server and Client) that Deadline fighters synchronizer now has, adding json encoding/decoding procedures as required and research/implementation of various HTTP requests on different platforms/languages.
- Helped the team narrow down AWS services that can be used for the project. Set up and configured AWS account when Jacky had to leave the team.
- Configured Git repo for the team. Functioned as the Git "expert" on the team, helping people deal with cherry-picking, merging during conflict, etc.
- Added logging feature to all applications.
- Formatted all code files involved to remove dead code, set uniform indentation etc.
- Led brainstorming sessions on technology selection, conflict resolution, server implementation etc.
- Major contributor to code reviews, commenting on functionality as well as best practices.
- Helped co-ordinate meetings regularly by following up with team members. Letao helped when Sivaranjani had other commitments.
- Helped collate and format both the preliminary presentation and final presentation. Work included proofreading, editing and formatting for better presentability.
- Helped team members with the installation and usage of LaTeX
- Helped the team come to consensus using Google Forms, Doodle poll, meetings etc., as necessary.
- *Working on implementing the conflict resolution scenario at the time of this document submission.*

6 Evaluation

6.1 Project Objectives

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

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	Yes
	Two or more files	No
Synchronize	All files in the server and local syn folder	Yes

Table 2: Current status of implementation

6.2 Team dynamics

We have worked together in harmony for the most part of the project duration. Team size reduced during the week after the Preliminary report due to conflict. This not only led to redoing of some work but also added work load on the existing members. But the team managed to keep its morale through the whole process.

One of the members had significantly less contributions compared to the rest of the team. This was amplified by lack of communication, attendance and general participation from the member. The gender skew of the team could be a possible reason for this.

Also, sometimes, there was difficulty in communication due to the fact that more than 50% of the team is Chinese.
The following is the distribution of 100 marks among members. Note that this was assigned with consensus of 4 members as the 5th member was absent on the day the marks were allocated:

Student Name	Marks
Georgii Fidarov	2
Letao Wang	24.5
Lili Chen	24.5
Qilin Zhou	24.5
Sivaranjani Subramanian	24.5

Table 3: Mark distribution on team consensus