Project Report

Project DIY Cloud

Ka-Him, HO

School of Computer Science & Informatics, Cardiff University

Supervisor: Ralph Martin

Moderator: Irena Spasic

Module: CM3203 One Semester Project (40 Credits)

Student Number: 1265987

May 5, 2015

Abstract

The aim of this project is to build a DIY cloud solution, based on an FTPS server controlled by the user. This report looks into the background, design, implementation and testing of creating a file synchronisation application using ftps protocol for Android devices and Mac OS X devices. The report also discusses problems I encountered, how the application can be improved in the future, and my reflection on the project as a whole.

Acknowledgements

Many thanks to Professor Ralph Martin for his valuable and constructive suggestions for this project.

Table of Contents

1. Introduction

1.1. Goals of project

The aim of this project is to build a DIY cloud solution, based on an FTPS server controlled by the user. Many individuals or companies have their own FTPS servers. By using our own FTPS server, we ensure data is not giving out without notice, but these are not applications, they are file transfer protocols, they do not synchronise data in background automatically, like Dropbox and Google Drive. Therefore, we need a tool that automatically synchronise data between the server and local computing device, like various cloud solutions do, in form of FTPS protocol controlled by users.

The client application will be provided in Android (Mobile device) and Mac (Desktop) platform. Users will be able to select folders to be synchronised to FTPS server. Any changes on the master copy on the server will download automatically in background in a time frame. Thus, any updates to local files will upload to the server automatically. There are several major concerns of this project, including, issues which clients going offline for a long time, conflicts if two clients change the file at the same time, battery usage on mobile devices, data loss during updating the file, ease of add and remove files, and ease of selection of folders to be synchronised.

This application will benefit users that want to use their own controllable FTPS server as the back-end of cloud solutions, who need to synchronise data among different clients.

1.2. Target Audience and Beneficiaries

The application targeted towards people with their own FTPS server who willing to synchronise files among different devices or synchronise local files to server as a backup. As there are many other cloud drive solution provided by big companies, I assume the targeted audiences are interested in privacy and security. People do not need to trade-off between the convenience of storage services that synchronise automatically in background and the concern about service providers giving out data to government [1]. Hence, users will not encounter data ownership problem with their own cloud solution, it is debatable that data uploaded to providers' servers own by which parties.

In addition, they should have some basic IT knowledge to understand how to find or set up their own FTPS server.

I would consider myself as a beneficiary as I improved many of my existing programming skills and gained a lot of additional knowledge of various topics such as programming with Java on Android platform and Object Orientated Programming.

1.3. Scope of project

In order to ensure I have a successful project, it is important to maintain focus on accomplish the aim and objectives. There are two completely different client applications using different programming languages. There are different approaches to each implementation steps and design decisions. The most important algorithm and theory in this project is the remote synchronise problem. It will be discussed in more detail in theory section. When I started my project, I chose to use different languages on different platform; as I thought it would be a wiser option to use technologies which I am comfortable in.

1.4. Assumptions

There are few assumptions throughout the project.

First, I assume the end product is a working prototype which means I am more concentrate on implementing features that fits the specification than visually appealing of the applications.

Secondly, I have to assume user have their own ftps server or able to find their way to get one. Thus, I assume users have both read and write rights to directories they assigned to synchronise.

Thirdly, the ftps server is assumed to be the central server sync across different client devices. The remote server should have the latest file stored. Users are assumed not to modify files on server directly.

2. Background

2.1. Existing Solutions

There are several cloud solutions exist these days, most of them use their own protocol and own encryption method to provide synchronization service, for example, Dropbox and Google Drive. Dropbox uses Python to write cross-platform desktop applications, with their own UI abstraction layer. Besides, there are some open source community driven solutions, such as SparkleShare [2], ownCloud [3] and Syncany [4]. SparkleShare and ownCloud both provide self-hosted file synchronisation solution using their own protocol. They are fully featured aim to replace Dropbox-like services as a whole. Unfortunately, they require own server set-up with their server application. Syncany tried a different approach, they attempt to do a client side application that able to use different type of own storage (FTP, S3, WebDAV, NFS, Samba/Windows file share, ...). Syncany is a better approach on balancing ease of use and security.

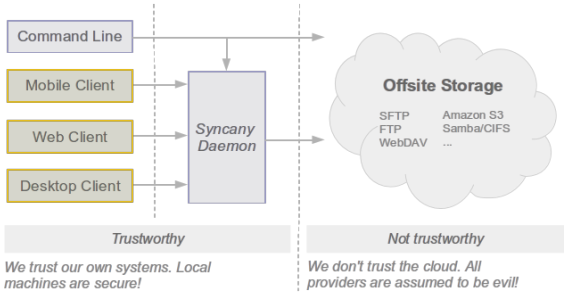


Figure 1. Syncany approach. [5]

On the other hand, there are some ftp client software capable to do file synchronization as an addition to their application. FileZilla and WinSCP [6] are both FTP clients, they include a folder sync feature hidden somewhere in their menu. Their solution is to do sync manually and check which file has a recent timestamps. The problem is their solution is not user-friendly, it is well-hidden and it requires manual trigger sync operation. In addition, an application called Fling does exactly FTPS synchronization; however, it can only run on Windows, and it can only upload and update existing file, files are not removable.

2.2. Theory

General applications that tend to do synchronisation between two directories, operate with the use of timestamps attached to files. First, I will talk about timestamps-based sync, then, I will discuss why the solution of numbering file versions is more suitable for this project.

The advantage of timestamps-based syncing is easy to implement and no extra cost of memory use, as only single property is used. However, time is a relative concept to different machines. Clock on client devices and server can be out of sync. Few seconds of differences may not cause a major problem, as systems can modify to allow seconds of time span. However, in some situation like different time zones or time zone with daylight time, if clock is out of sync it may cause significant problems. In these conditions, any algorithms that use timestamps as the key component, they may not work as intended. This problem can solve by generate timestamps on a single machine and compare to the one on another machine. While this solution may lower the impact on clock out of sync, but this causes single point of failure, as it depends on whatever generates the timestamps severely. In addition, timestamps-based solution does not scale well with multiple clients and it does not handle situations where client update file at the same time.

Alternatively, I choose to adopt a solution using version number for each file as a logical clock. This approach uses version numbers as logical clock is similar to vector clock sending messages in distributed system [7]. It does not rely on accuracy of the file system or device clock, this avoid clock out of sync situations or client and server in different timezone. In addition, it prevent single point failure to depends on one file attributes; the most recent file version is determined based on a combinations of attributes, file size and timestamp, instead of just file last modification date; if there is no last sync list, the system will ask user's decision. Hence, to ensure safety of users data, when there is uncertainty about which file is the most recent one, and when version of file is in conflict, the system will prompt user confirmation. Each client is independent, plus, in each file they have their own version count. Saved directory lists store information about deleted files; therefore, removed files from one location can be identified, instead of assuming not allocate files mean removed.

However, this solution requires memorise list of last directories with all file versions, which means extra memory usage as either database or a file is needed to store the list. One major concern about the directory list, is the list will keep growing, even all files are deleted; As removed files will mark as deleted, but not remove this entry from the list. A reset list function can slim down the list, though no one can ensure that all clients remove that entry at the same time. On top of the directory list, each file version object contains all clients with that file, which means client list also keep growing; if there are lots of clients attempt to sync the same directory, it may slow down the performance. On the other hand, this solution unable to identify file rename, these files will treat as delete

old file and add a new one. Moreover, neither timestamp-based sync nor sync using a version object is really related to the content of what is being synced.

Comparing Version Objects

- Each file has a version object, each object contains all clients with their file version.
- When comparing two version objects:
    - If there are equal,
        - with equal file size and date, then Skip this file.
        - with different file size or date, then file conflict.
    - If client side has a larger value than server side, file on client side is the one more updated.
        - If mark as deleted *(file size and date = -1)*,
          Remove file on remote server.
        - Otherwise,
          Upload this file from local to remote server.
    - If server side has a larger value than client side, file on server side is the one more updated.
        - If mark as deleted *(file size and date = -1)*,
          Delete file on local client.
        - Otherwise,
          Download this file from server to local.
    - Otherwise, this file is in conflict
        - If mark as deleted *(file size and date = -1)*,
          Remove file on remote server and Delete file on local client.
        - Otherwise,
          Let user to choose which file is the latest one.

3. Specification

3.1. Functional Features

The aim for this project is to create an application that automatically synchronise files between client and FTPS server where directories to be synchronised are controllable by the user. There are few requirements set in the initial plan, and below is the list of requirements set after the final consideration.

1. The application should store a master copy of selected files on FTPS server.
2. The application should do file exchange and connection using FTPS protocol.
3. The application should download updates when master copy on the server changes.
4. The application should upload files when local copy changes
5. The application should run in the background without command synchronous action manually or any user interaction after the initial setup.
6. The application should handle conflicts like updating the same file at the same time.
    - 6.1. The application should let user to select which files is the latest one when conflict occurs.
    - 6.2. The application should let user to choose whether they want to select each files when conflict occurs or set a default action.
7. The application should handle users that offline for a long time of period.
8. The application should consider battery usage on mobile devices.
    - 8.1. The application should give users control on synchronous in certain condition. I.e. in mobile network or/and in Wi-Fi network.
    - 8.2. The application should give users control the interval of check time. I.e. able to custom check every X seconds or minutes.
9. The application should avoid data loss while updating local and server's data.
    - 9.1. The application should not delete any files before it ensure files updated successfully.
10. The application should be easy to select or remove directories to be synchronised.

10.1. The application should implement a folder picker, instead of forcing user to enter the complete path manually.

11. The application should be easy to add or remove files in selected synchronize directory.

12. The application should be implemented on both Mac (Desktop) and Android (Mobile) platform.

Additional Features

- The application should store a list of log to save a record of transfer actions.
- The application should let user to choose to enable auto schedule sync or not

3.2. User Interface

Apart from the functional requirements, there are some conditions on user controls.

1. Users should be able to add or edit selected sync directory.
2. Users should be able to trigger sync operation manually, by pressing the button.
3. Users should be able to start sync operation automatically with a given time frame.
    a. User should be able to set the sync interval.
    b. User should be able to switch auto sync on and off.
4. Users should be able to clear logs and selected sync directory.
5. Users of Android application should be able to restrict the application to perform in certain network environment and in certain battery status.
    a. Users should be able to set these in preferences.
6. Users should be able to cancel or stop current progress.
7. There should be a log screen with all transfer details in it.
8. There should be a sync path list with a button to add new entry.
9. There should be a dialog box show up when pressing add button.

10. There should be a folder picker in dialog box for adding or editing selection of sync directory, instead of requiring users to enter path manually.
11. Notification should show up on every steps taken, for examples, uploading files, getting the latest list of directories.
12. Notification should show up when there is any error message regards to the sync process.
13. Users should be able to tell auto sync is enable on screen,
14. Users should have the control which file is the latest one when sync process encounter problems.

4. Design

4.1. Flowcharts

Flowcharts are an effective way of explaining how specific algorithms run. They can often be more useful than words. Most of my code in two different platforms are based on the following flowcharts.

Add/Edit Sync Directory

Adding and editing selection of sync directories are using the same flow, as they are basically the same thing and sharing the same code. At the beginning of the flow, if it receive defined local and remote path as input data, it will consider as editing; otherwise, it will consider as adding.

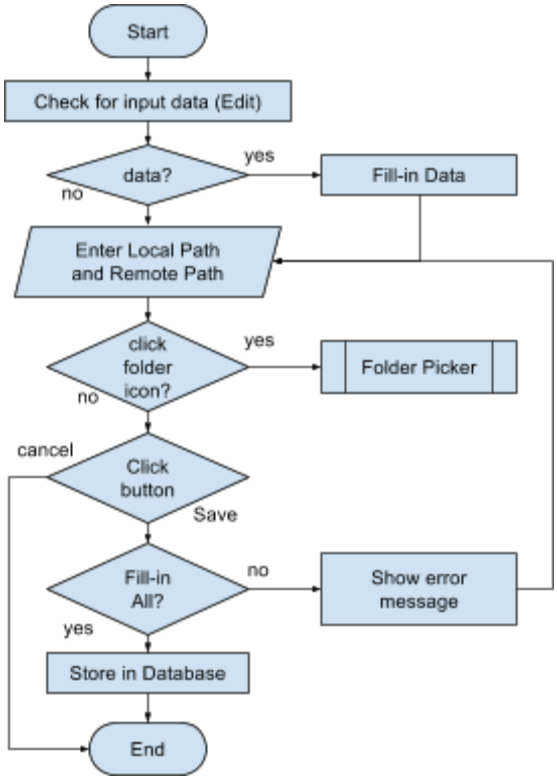


Diagram 2. Flowchart for adding or editing sync directory.

Sync - main flow

As mention in the theory section, the synchronization method used in this project requires directory lists to do comparison; this solution avoid to single attribute, file modification date, as the key factor.

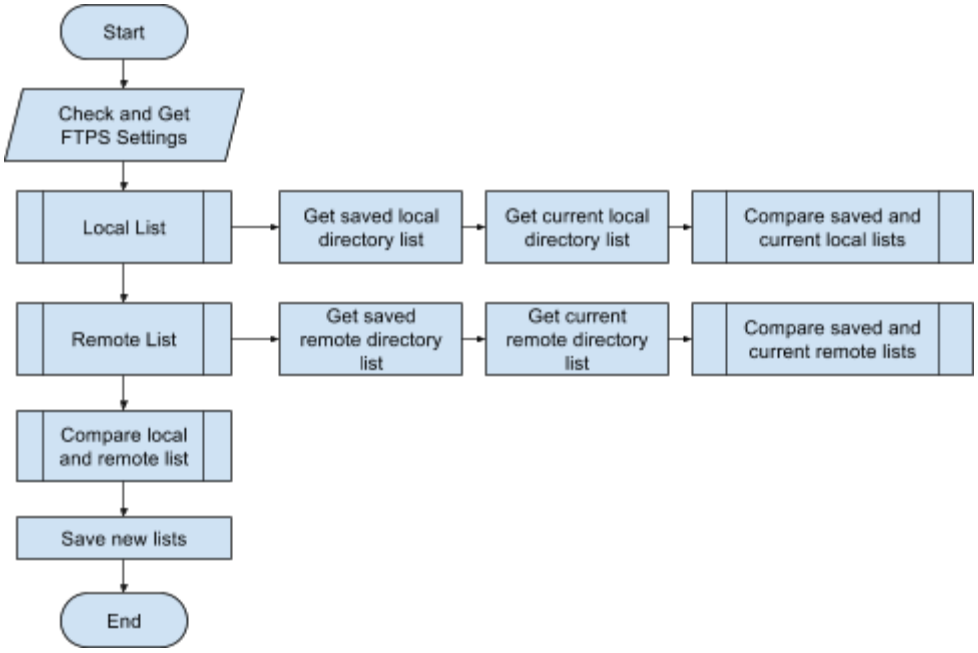


Diagram 3. Flowchart for main flow of synchronisation

Sync - Compare previous and current list

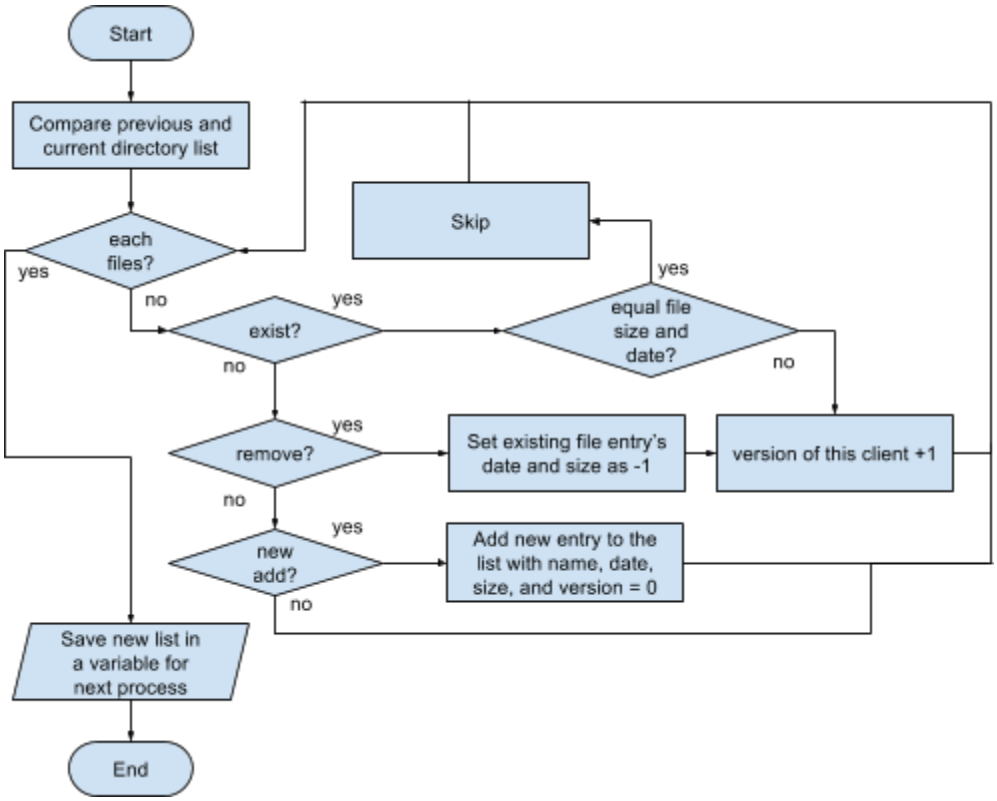


Diagram 4. Flowchart for comparing current list and previous list in sync process.

Sync - Compare local and remote list

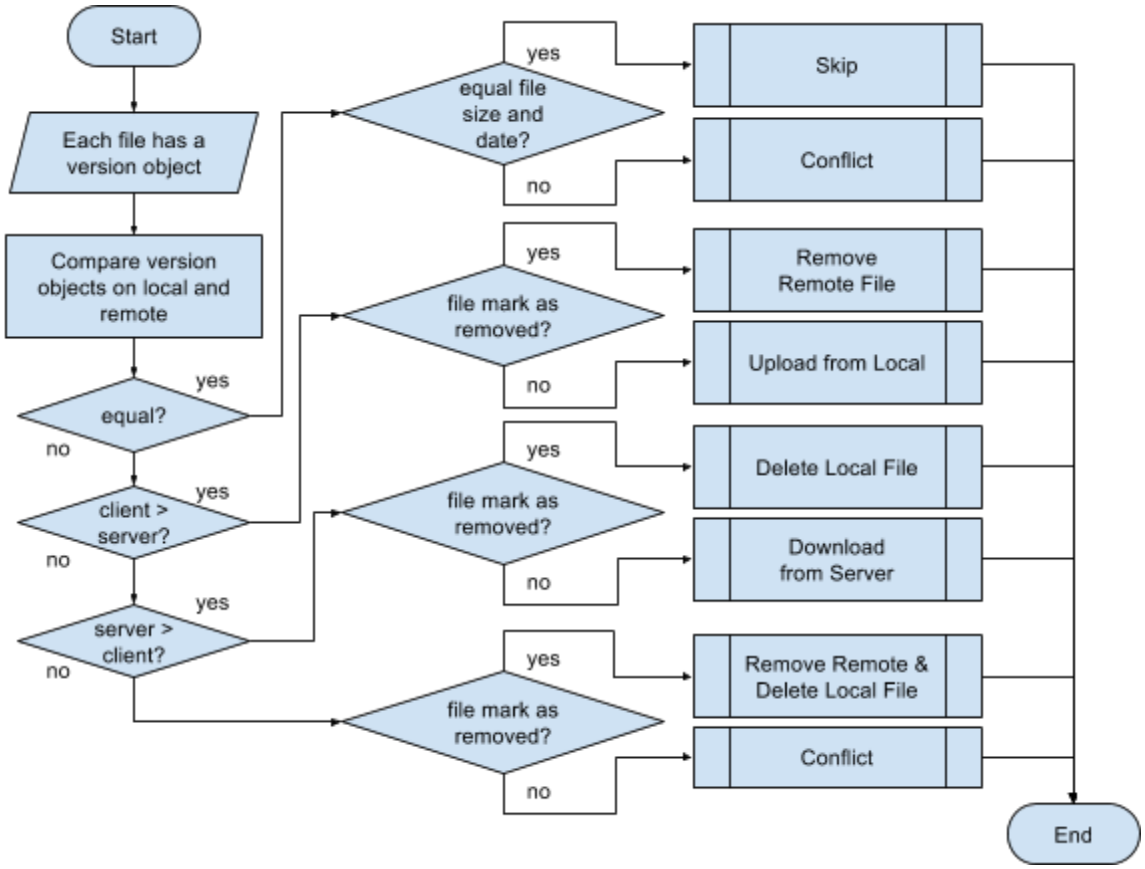


Diagram 5. Flowchart for comparing local list and remote list in sync process.

4.2. Class Diagram

Full version of the class diagram is in Appendix A, following diagrams extract some significant functions and classes of the Android application.

The *SavedList* is a group of classes that handle to directory list. Every time at the beginning of synchronisation, the system will get the latest list and compare with the saved list; this comparison performs using compare() method in *SavedListUtils*. *SavedListUtils* contains all common methods used by *LocalSaved* and *RemoteSaved*, such as comparing list, add file entry to the list and mark file entry as removed.

RemoteSaved is using FtpsUtlis to operates all ftps connections, this class is used to formulate the results to a group of methods similar to *LocalSaved*. This approach helps to organise two types directory list and ease the comparison between them.

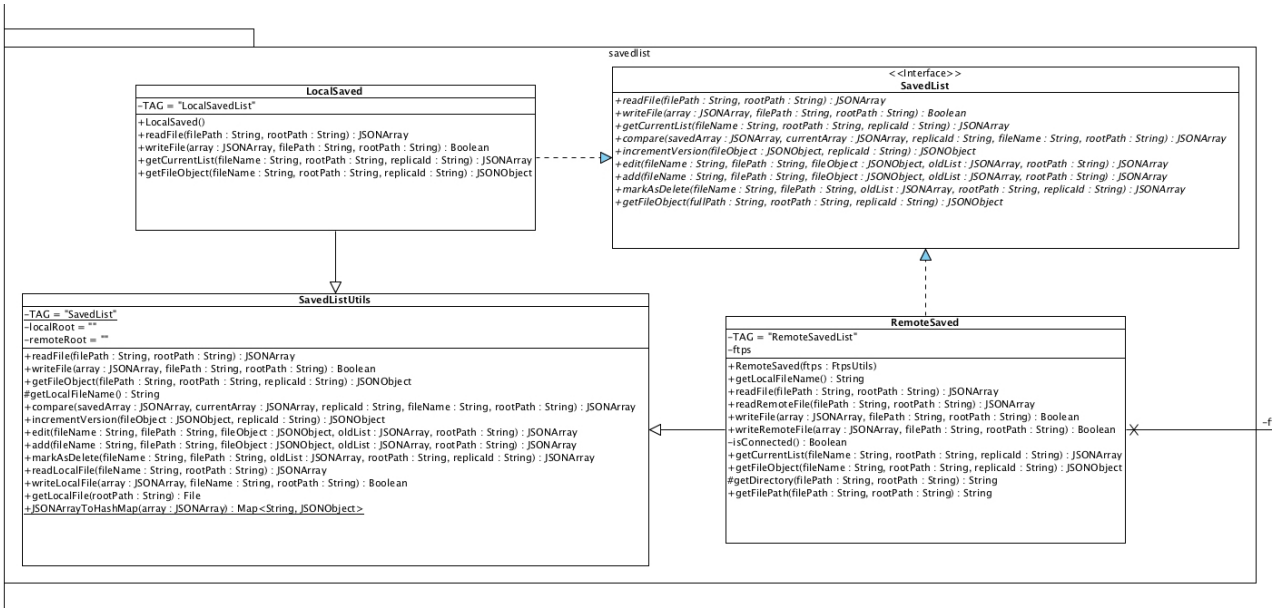


Diagram 6. Class Diagram for SavedList, classes for saved directory list

FtpsUtlis is a class responsible for all kinds of ftps connection, like connect to server using ftps protocol, retrieving directory list, and uploading and downloading files.

It has method to get the latest list of directories called by *RemoteSaved*; the list will store in a *FileTranferHelper* object. It also has methods to read and write the list from the server. Furthermore, it has a method to provide directory information in a *FTPFileHelp* object, to FTPS folder picker in Add/Edit Sync Directory activity.

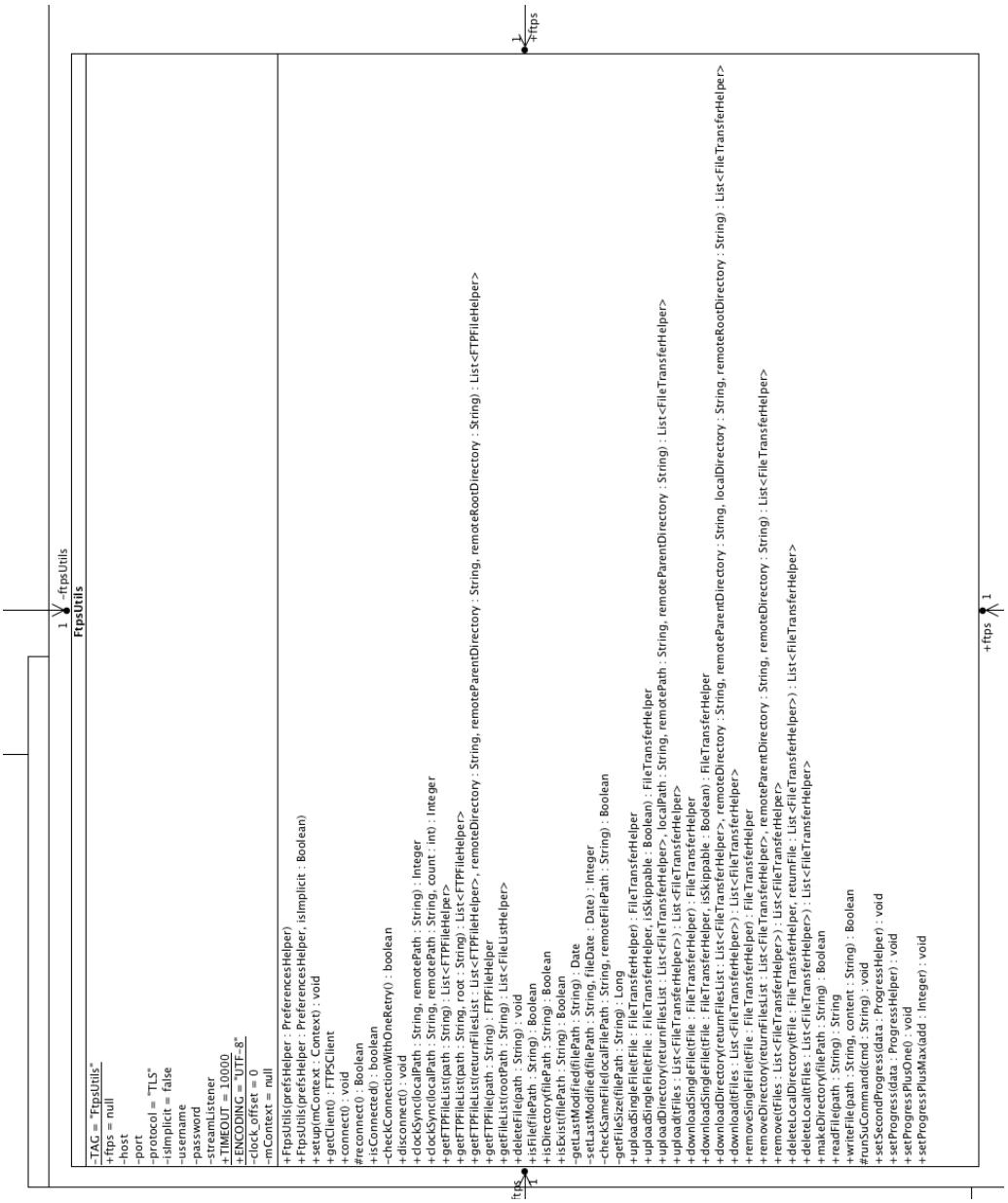


Diagram 7. Class Diagram for FtpsUtils.

Below is a group of classes for synchronization and services, including FileTransferHelper, TransferService, ForegoundService, SyncService, PreferenceHelper and SyncSelectActivity.

FileTransferHelper: It contains data of transfer files, like filename, size, date, transfer status. It used for different classes to communicate the transfer status. *TransferService* uses details from this class to perform data connection with the server. *ForegroundService* uses details from this class to insert transfer status to log database. *SyncSelectActivity* uses details to display file information and perform transfer command after user selection. Communication between classes perform under broadcaster and receiver.

TransferService: an IntentService to do transfer file actions, such as upload, download, delete local and remove remote files. Then record the action to log database. This class call *FtpsUtlis* to perform certain tasks.

ForegoundService: a Service that handles the sync interval trigger and receive manual sync command. This is the class the responsible to show notification. Notification will show according the message received from broadcaster. Any transfer actions and status will send broadcast within the app to notify their status.

SyncService: an IntentService that does all comparison on directory lists and send transfer command to *TransferService*. When there is more than one sync directory selected, this service acts as a work queue; IntentService provides such work queue function.

PreferenceHelper: It requests saved preferences and formulate into a class object to let other classes to call specific item in preferences.

SyncSelectActivity: a pop-up dialog box interface for selection of file when sync conflicts happens. It receive a *FileTransferHelper* object that contains file details from broadcaster. When users made their decision to choose either local file or remote file, the transfer command for either upload local file or download remote file will broadcast; then, *ForegroundService* will receive the command and call *TransferService* to operate the command.

Diagram 8. Class Diagram of Synchronization and Services.

4.4. User Interface

4.4.1. Draft Design

Before the start of implementation and final design, I drew draft design on stretch to give myself a clear idea on how user interface should work. Below section is some of my drawing on different functions and screens. These figures focus on Android application.

Main Screen

In the main screen, there will be few buttons, including the setting button to redirect to setting page, and the manual sync button for user to trigger sync action manually. In addition, a progress bar to show status of synchronisation. Next, it the log section at the bottom of the screen

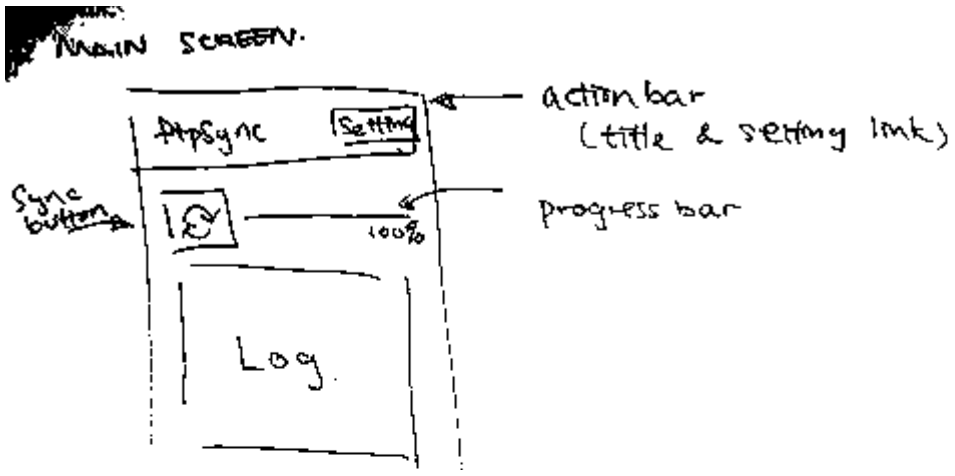


Figure 2. Draft Design - Main Screen

Sync List

In the list of synchronise directory, each entry will show the local and remote path. In addition, an icon to indicate their status. When clicking the entry, it will open the edit dialog box. While long-pressing a menu will pop-up with options to edit or delete the entry.

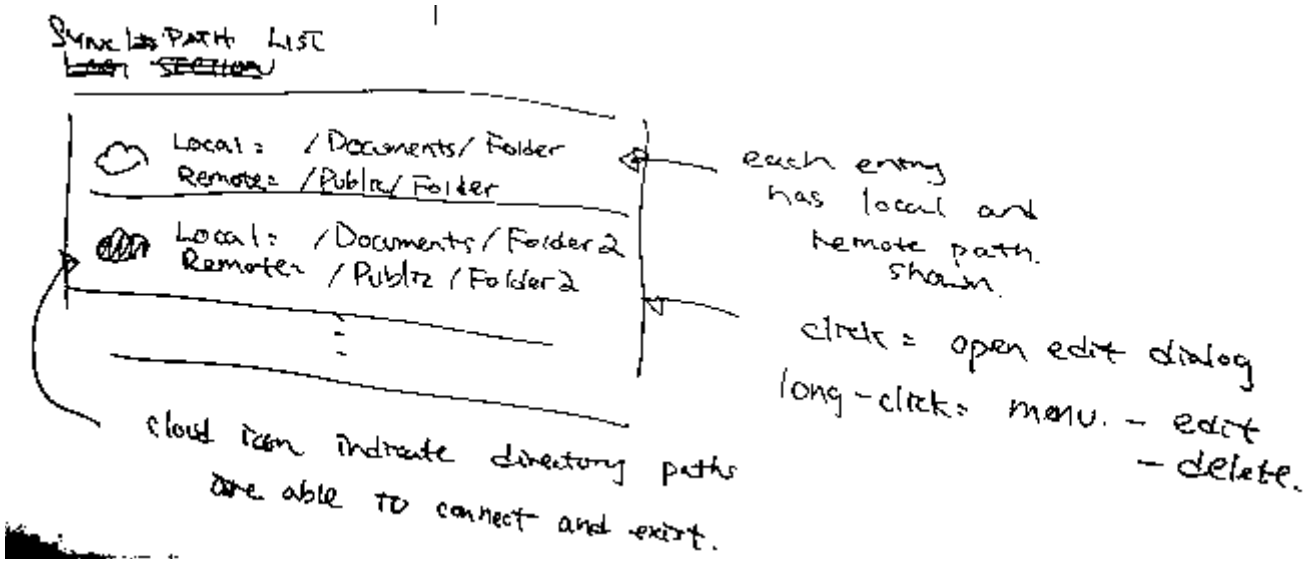


Figure 3. Draft Design - List of Sync Path

Log Section

The log section includes all transfer actions like upload, download, delete local file, remove remote file and skip. Each log includes the action, time of the action, local and remote file path, and transfer status (Done, Skip, False). When long-pressing the entry, a menu will pop-up with the option to delete that entry. A clear all button will be included at the bottom of the section.

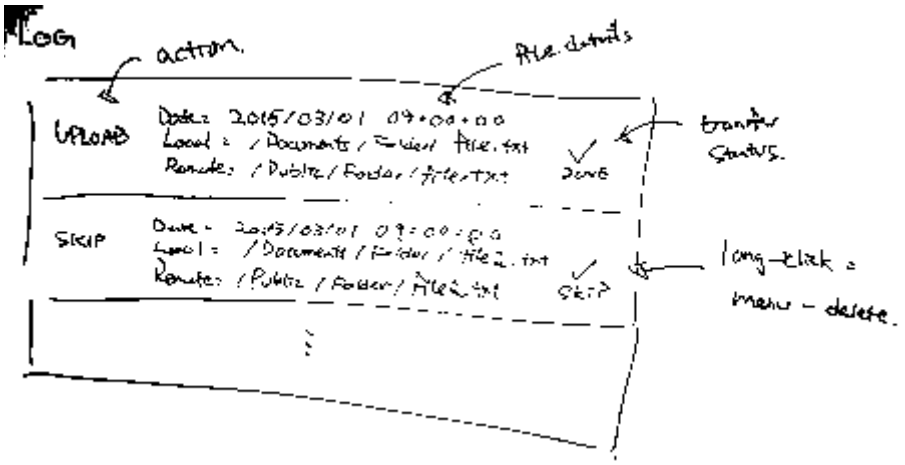


Figure 4. Draft Design - List of Logs

Add / Edit Sync Path Dialog Box

The dialog box for adding and editing sync directory, include, text entries for user to enter local and remote path. In addition, there is a folder button to call folder picker. The folder picker let user to select specific folder with a graphical interface, and return a full path of specific directory. If user click cancel, none of the changes will be saved.

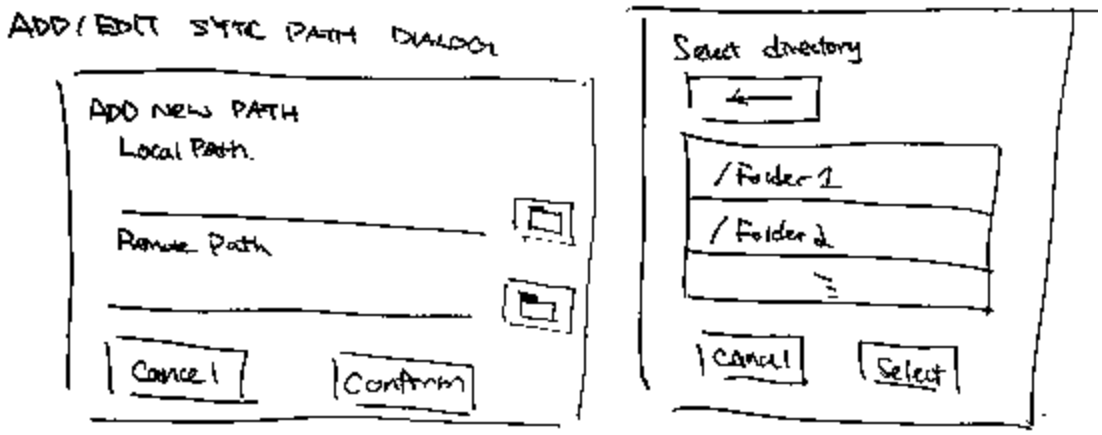


Figure 5&6. Draft Design - Add/Edit Sync Directories Dialog, and Folder Picker Activity

Solve Conflict File Dialog Box

This is the pop-up dialog box when file conflict occurs during synchronization. This box let user to select either local file or remote file. If user choose local file, the application will upload the file from local client to remote server. Otherwise, if user choose remote file, the application will download remote file from remote server to local client device.

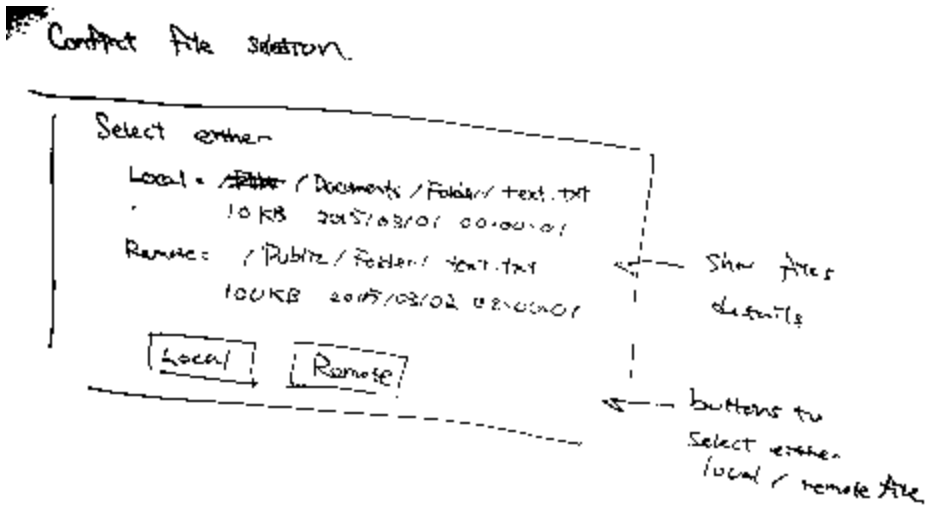
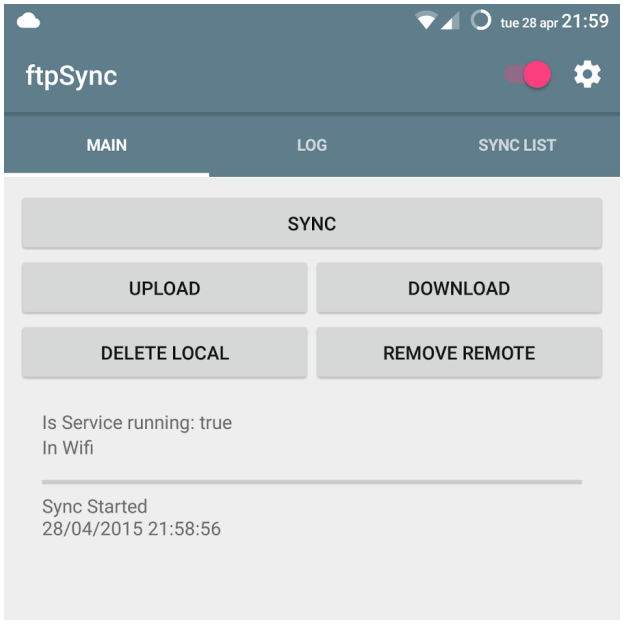


Figure 7. Draft Design - Conflict file selection dialog

4.4.2. Final Design - Android

Main Section

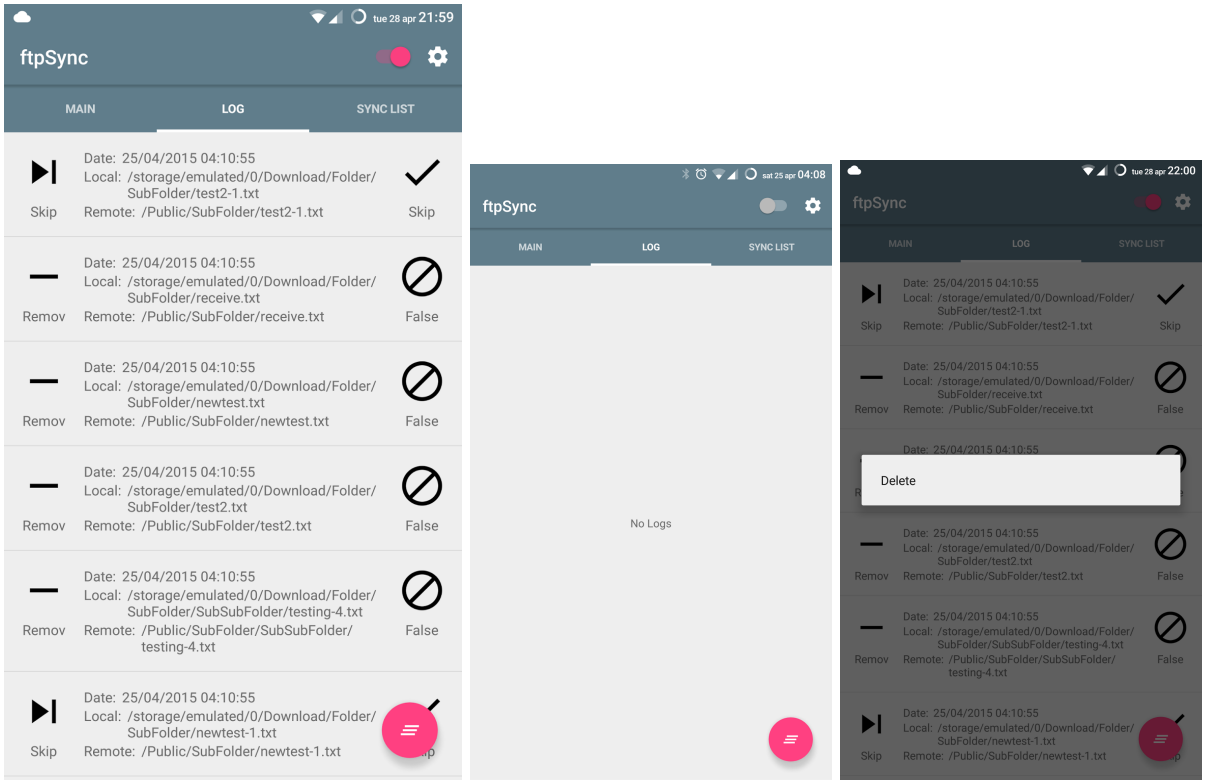
In the main screen, there are few buttons, the most important ones are Preference Button that redirect user to settings page, and Manual Sync Button to start sync manually. The switch button at the top in action bar is the switch for foreground service to enable auto sync; this gives users easy access to control the foreground service. In addition, at the bottom of the screen, there is a progress bar to show the status of synchronisation. As this is a prototype of the tool, I placed four different action buttons on screen in order to test each transfer operation. In future, this screen can be removed; notification is capable of showing exact progress messages; those four action buttons are not necessary, they could hidden in context menu of sync list when long-pressing entry; and the manually sync button can move to action bar.



Screenshot 1. Final Design - Main Screen

Log Section

In the draft design log section is under buttons and progress bar, after consideration, I chose to make log a separate log screen. The log screen includes all transfer actions. Each log includes the action, time of this log record, local and remote file path, and transfer status; Icons are used to provide clear indication of each entry. When there are no logs in the database, “No Logs” message will be shown; therefore user will able to tell the application is working properly, instead of an empty screen with no indication and informative feedback. When long-pressing each entry, a context menu will pop-up with options to delete that entry; this action placed in long-press menu, as people tend to clear all logs at once in normal situation. On the other hand, a clear all button is at the right bottom corner of the screen; user should be familiar with this button as they are part of the android design guideline.



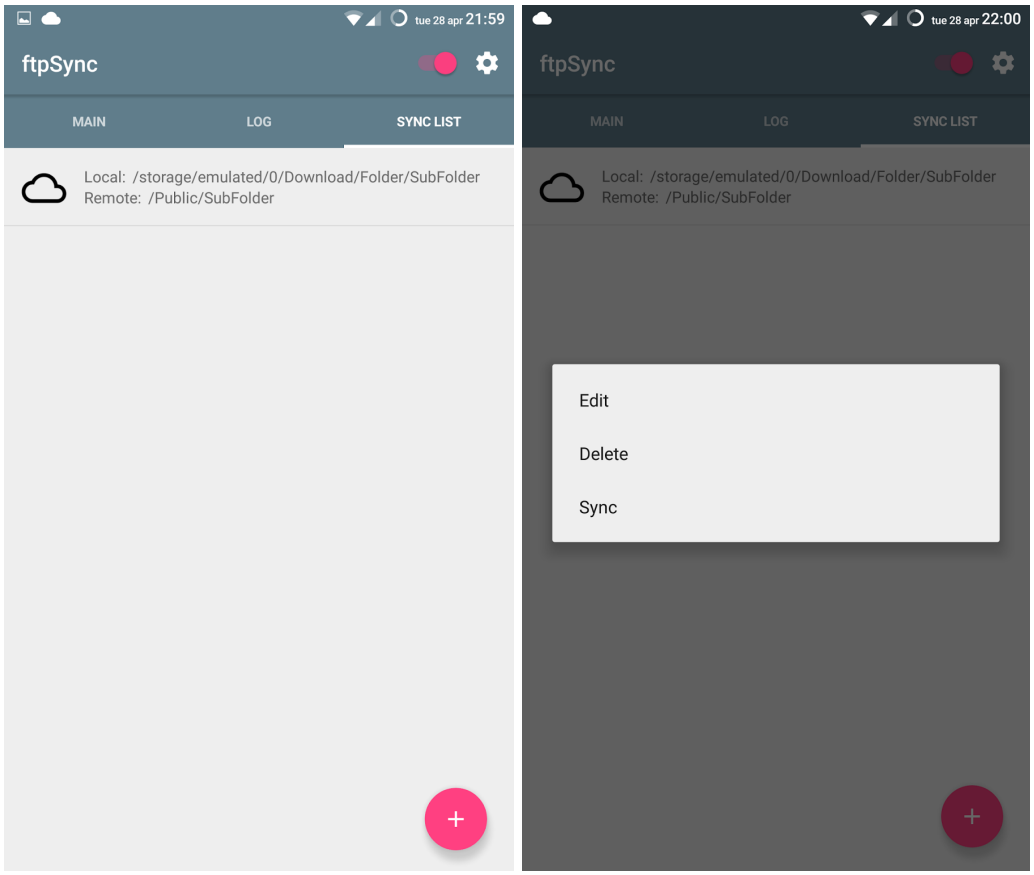
(Left) Screenshot 2. Final Design - Log Screen

(Center) Screenshot 3. Final Design - Log Screen with No Logs

(Right) Screenshot 4. Final Design - Log Screen when long-press entry

Sync List

In the list of synchronise directory, each entry is showing the local and remote directory path. Besides, due to user's expectation, a response message will be shown to indicate there is no sync path in the database. When clicking the entry, it will open the edit dialog box. While long-pressing a menu will pop-up with options to edit or delete the entry. As in normal circumstances, users are more likely to edit their path rather than delete the entry; in addition, this long-press action provided an extra precaution where users accidentally remove an entry. At the bottom of the screen, there is a add button (“+”) which let user to add new entry, a pop-up dialog box will be shown; this implementation of floating action button (FAB) is one of the material design guideline, it may help users to reach this action easier, as the other option to place this action button is the action bar at the top of the screen.

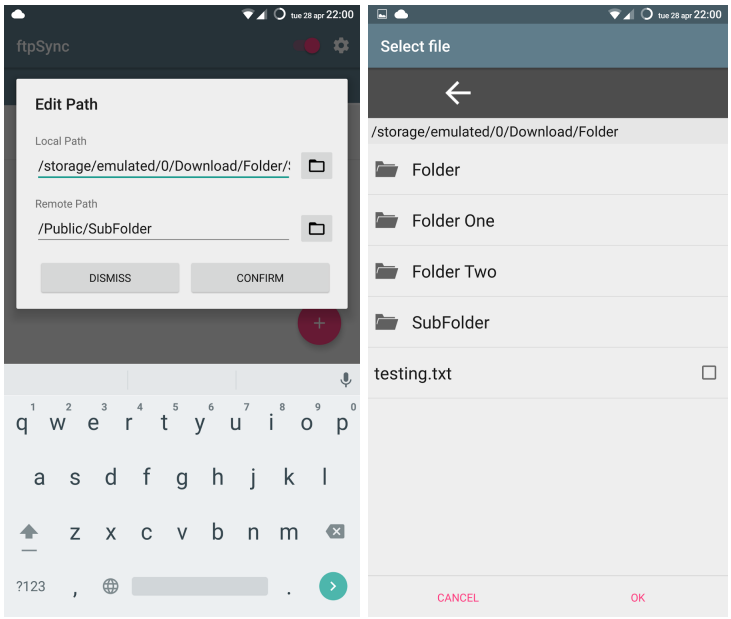


(Left) Screenshot 5. Final Design - Sync List

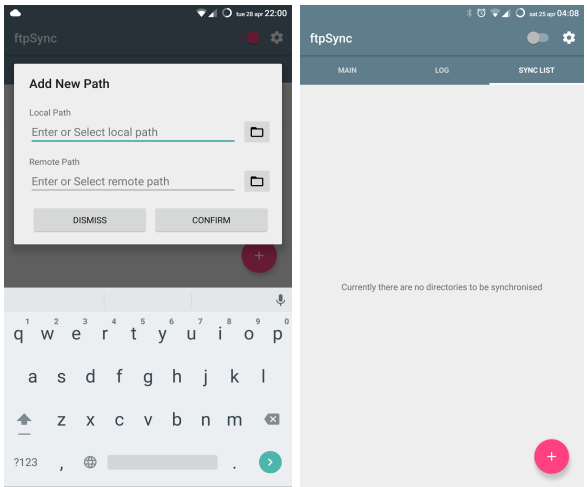
(Right) Screenshot 6. Final Design - Sync List when long-press entry

Add / Edit Sync Path Dialog Box

The dialog box for adding and editing sync directory, include, text entries for user to enter local and remote path; experienced users can enter the path manually for a faster process. In addition, there is a folder button to call folder picker for novice users; when there is a path in the text entry, the folder picker will try to open that directory first; this attempt to help users to check particular directory exists or not, especially those enter path manually or edit entry after directory removed somewhere else. The folder picker let user to select specific folder with a graphical interface, and return a full path of specific directory. If user click dismiss, none of the changes will be saved.



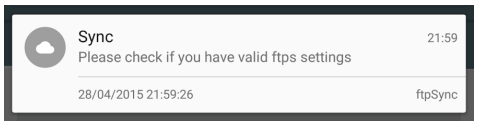
(Left) Screenshot 7. Final Design - Edit Sync Directory Dialog Screen
(Right) Screenshot 8. Final Design - Folder Picker (using framework)



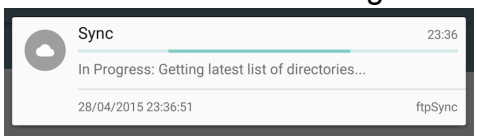
(Left) Screenshot 9. Final Design - Add Sync Directory Dialog Screen
(Right) Screenshot 10. Final Design - Sync List when no directories set

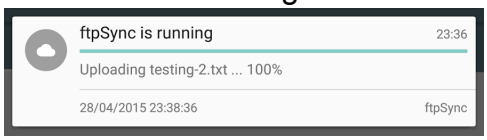
Notification

In addition to progress bar and status messages in main screen of the application, a notification will be shown when foreground service started. Below screenshots show different situations, including when user not having a valid FTPS settings, when getting the latest list directories, and when uploading file when progress stated. Notification is the best way to notify users certain action performed and some error occurs, without disturbing other task performed by the user or requires user to enter the app to get progresses.



Screenshot 11. Final Design - Notification when having invalid FTPS Settings

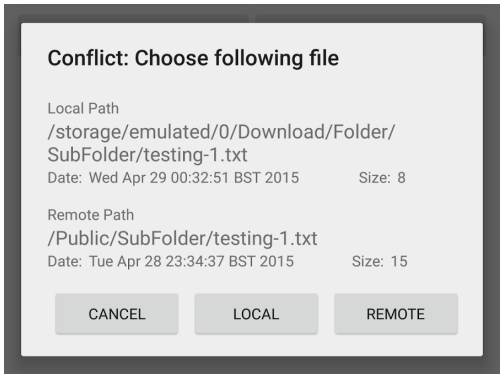




(Left) Screenshot 12. Final Design - Notification when getting and parsing directory list
(Right) Screenshot 13. Final Design - Notification when uploading file

Solve Conflict File Dialog Box

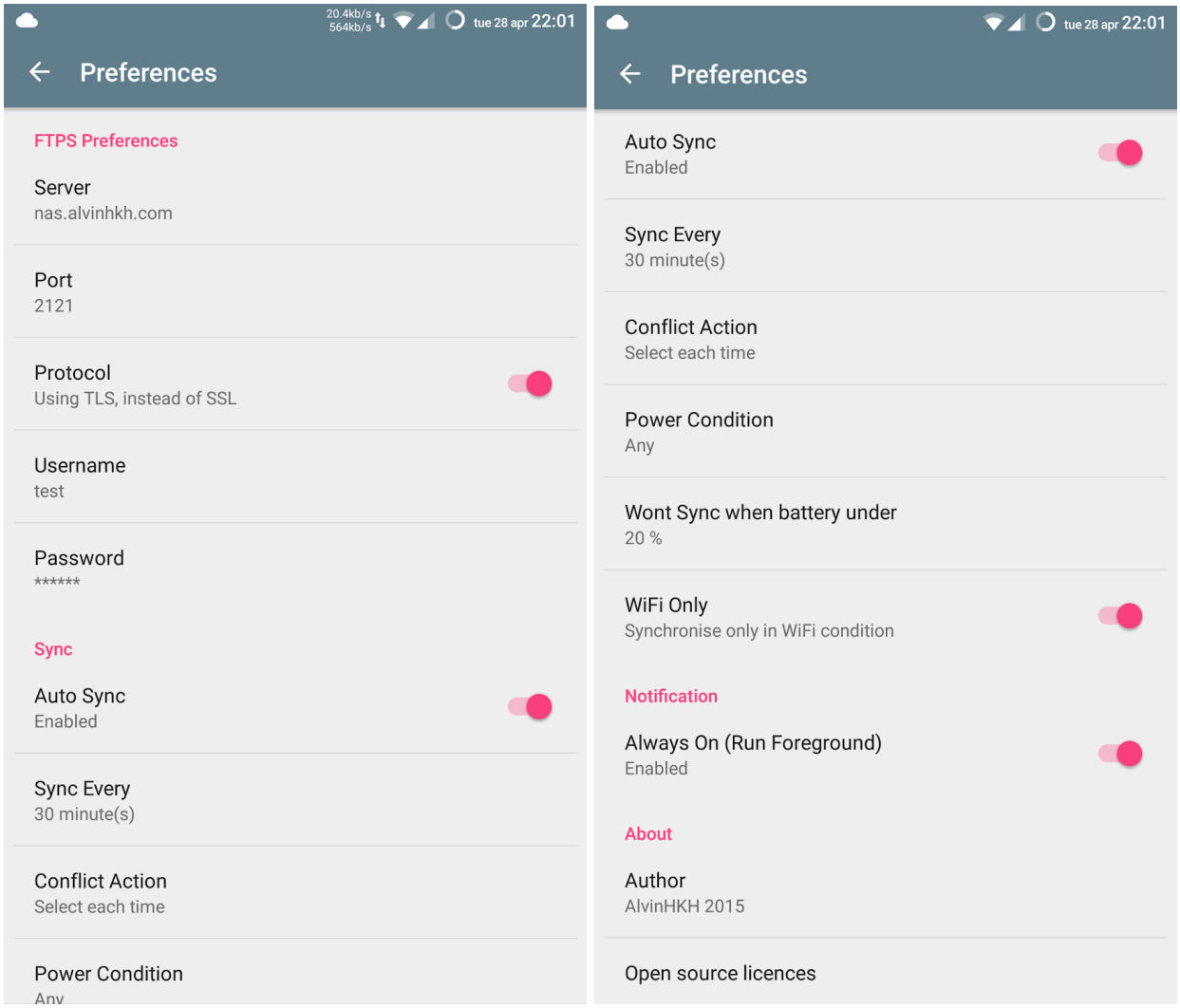
Whenever file conflict occurs during synchronization and user chosen to solve conflict each time occurs, this dialog box will pop-up. These box contains all information needed to let user to decide they want to keep either local file or remote file. If user choose local file, the application will upload the file from local client to remote server. Otherwise, if user choose remote file, the application will download remote file from remote server to local client device. This dialog box will only show up if user choose to solve conflict files each time occurs, alternatively, users can set default action in preferences. In the future, I might transform this selection dialog box to expanded notification; therefore, users may have the control of solving conflict files but no pop-up boxes to interrupt the screen.



Screenshot 14. Final Design - Conflict File Selection Dialog

Preferences

The preference screen is split into four sections, FTPS Preferences, Sync, Notification and About; I split preferences into different sections, so users have a clear message where each part means certain part of the application. The ftps preferences section contains all settings about connecting ftps server. Sync section contains settings to enable auto sync, set sync interval, choose the default action when file conflict occurs, sync only in specific power condition and battery status, and sync only when user connect to WiFI network.

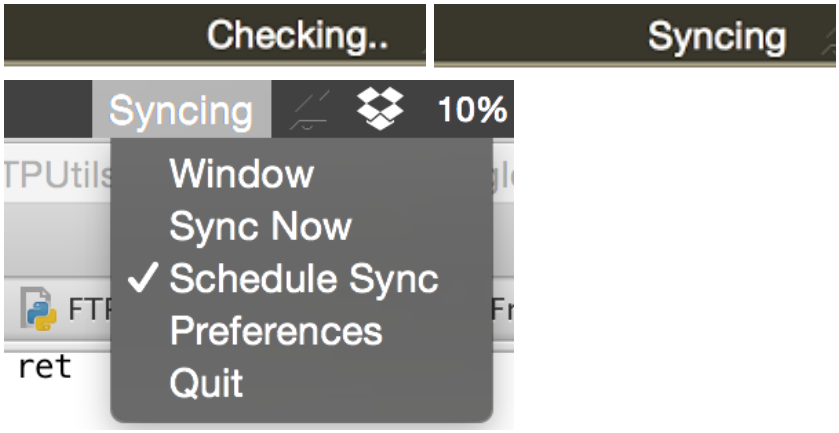


Screenshot 15. Final Design - Preferences Screen

4.4.3. Final Design - Mac OS X

Status Bar

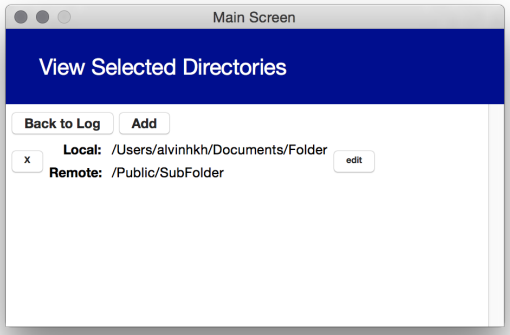
In Mac OS X version of the application, the main navigation among different screens and functions is in the status bar. The menu in status bar contains “Window”, “Sync Now”, “Schedule Sync”, “Preferences” and “Quit”. “Window” uses to open sync list, add item to sync list and log screen. “Sync Now” trigger sync action manually. “Schedule Sync” is the option to enable sync automatically every x minutes. “Preferences” uses to open setting window. As the application needed to stay in background to perform tasks, and user interface needed to place somewhere on the devices. Instead of an ongoing window that always on the screen; This approach of using a status bar provides users a clean and easy way to navigate around the application.



Screenshot 16. Final Design - Mac Status Bar Menu

Sync List

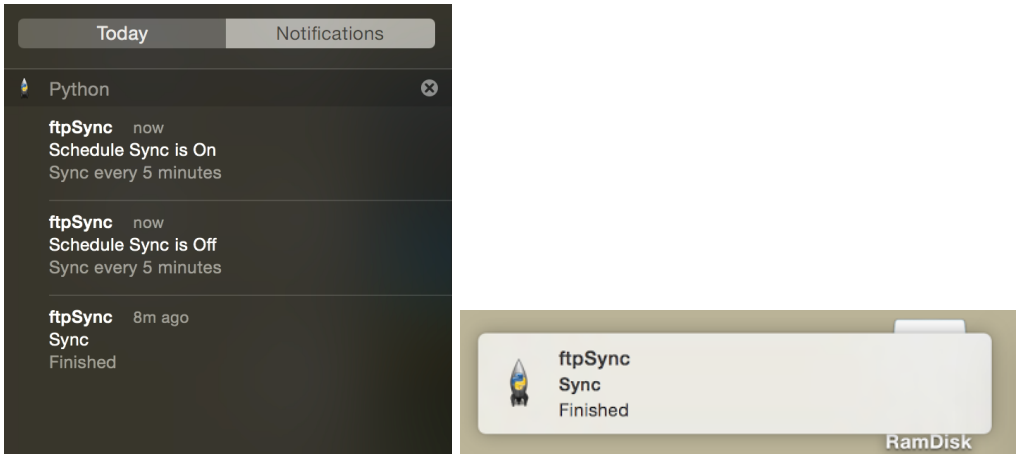
In the list of synchronise directory, each entry is showing the local and remote directory path. There are two buttons for each entry, “x” to delete that entry, and “edit” to edit that particular entry; due to the fact that desktop is not a touch device, I do not use the same approach as Android application to hide actions in context menu; instead, I placed buttons along with the entry. Besides, there is a “Add” button at the top of the screen to navigate to add new entry screen.



Screenshot 17. Final Design - List of Sync Directories

Notification

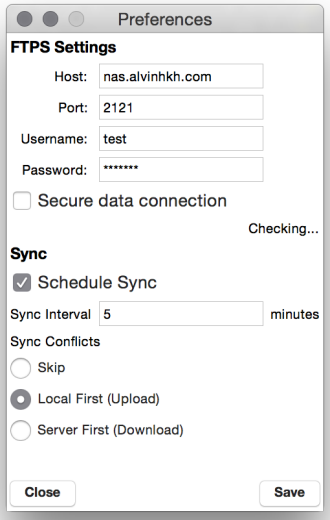
Notification will be shown when sync started, finished, schedule sync is enabled and disabled. It will also notify user they do not have valid ftps settings. By using rumps module, the application provides a native experiences on showing notification when sync started and finished operating.



Screenshot 18. Final Design - Notification in Mac App

Preferences

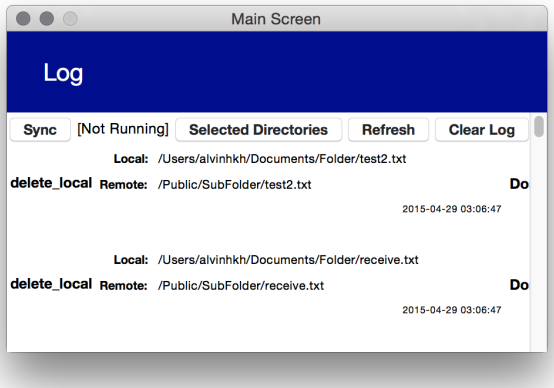
The preferences window is split into two sections, “FTPS Settings” and “Sync”. The ftps preferences section contains all settings about connecting ftps server. Sync section contains settings to enable schedule sync, set sync interval and choose the default action when file conflict occurs. In addition, whenever users open the preferences window or updated settings, the application will check the connection with defined ftps server, to provide a clear message of server availability.



Screenshot 19. Final Design - Preference Screen in Mac App

Log Screen

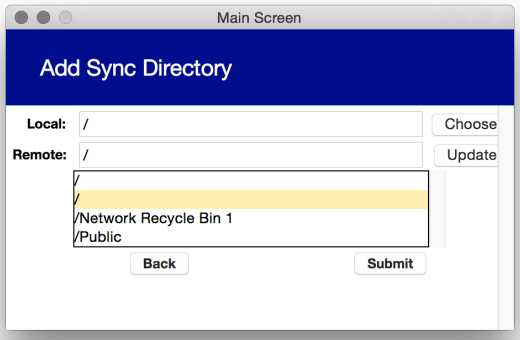
The log screen is the first screen after pressing “Window” in menu bar, as this is the screen users should view most, comparing to add or edit selection of sync directory. The log section of Mac app is similar to Android app, they both include all transfer actions. Each log includes the action, time of this log record, local and remote file path, and transfer status. Logs can be cleared at once by clicking “Clear Log”, but in this version log cannot be removed individually. A manual sync button is also placed at the navigation section, along with the sync status.



Screenshot 20. Final Design - Log Screen in Mac App

Add / Edit Sync Path Dialog

The window for adding and editing sync directory, include, text entries for user to enter local and remote path. For local directory, there is a “Choose” button and native selection window will pop-up. While for remote directory, user can select using the scroll box on screen. The “Update” button is for user to refresh the list of remote directory box. If user click back, none of the changes will be saved.



Screenshot 21. Final Design - Add Sync Directory Screen in Mac App

5. Implementation

5.1. Tools and Framework

It is crucial that tools or frameworks are selected carefully, as choosing the right tools will lead to successful project outcome and may help implementation a bit smoother.

5.1.1. Android Development

For Android development, I used the Android Studio (version 1.2 beta 3) with gradle support. Android Studio is developed by Google based on IntelliJ. It has a better graphical interface for theming android application. Besides, it is easy to debug the application via adb connection. Android Studio uses Gradle build system, it allows developers to add external libraries and frameworks when building and compiling the application. It makes adding these extensions easier, as developers like me no longer require to download libraries manually and add to the project, compare to the way that Eclipse does. I find it practically useful is Android Studio (IntelliJ) has a powerful refactoring support and Java auto-completion.

I have used two external libraries integrate to the Android application of this project, “FloatingActionButton” by makovkastar (version 1.3.0) [8] and “NoNonsense-FilePicker” by spacecowboy (version 1.1.3) [9]. These libraries are open source and I credited them within the application in the preference screen. In addition, I extended the file picker to support listing FTPS directories and files using the class written to connect FTPS server in this project *(FTPUtlis)*. I chose to integrate these libraries to the project as they are easy to adopt and suitable for the application.

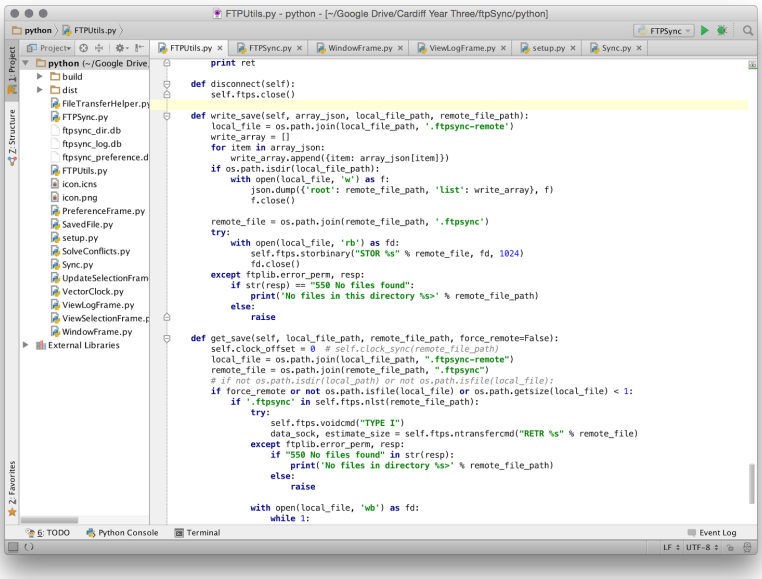
As this is my first time in coding on Android platform, I tend to think I should put more effort on implementing the core function, file synchronization, instead of putting time on developing elements that other developers had implemented and shared publicly. Besides, in order to match the material design language introduced by Google in Android 5.0, I used the material design icons package provided by Google. [10]

5.1.2. Mac OS X and Python Development

On the other hand, when developing Mac OS X version of the application, I used PyCharm CE by JetBrains, the same organisation who develop IntelliJ. They have a similar user interface and same powerful refactoring support and Python autocomplete. It has on-the-fly error checking and keeps code quality under control with PEP8 checks, therefore I can less worry code standard like naming variables and methods. Although in this project I was developing on two different platforms using two different languages, I used the same IDE for different platform with a similar programming environment.

In terms of modules I used in the Mac OS X version of the application, there are two external modules and few default modules. In this project, I used *rumps*, **R**idiculously **U**ncomplicated **M**ac os x **P**ython **S**tatusbar apps [11], to generate a simple launch menu that feels native to Mac OS X system. The module shorten code require building status bar application in Mac OS X and Python. Besides status bar, it has native Mac notification support which fit the use cases of this project. Another external tool I used in this project, is *py2app* [12]. py2app is a Python setuptools command which will allow me to make standalone application bundles and plugins from Python scripts. It makes packaging python scripts into Mac OS X application easier.

For the core functions of the applications, I used *Tkinter* module to build graphical interface for all windows like log screen, preferences window, sync list screen and add sync directory window. In addition, I wrote a class to connect FTPS server using *ftplib* module. *shelve* module is used as the database of the application, logs and sync list are stored using this module.



Screenshot 22. Use of PyCharm CE to develop Python application.

5.1.3. General Tools

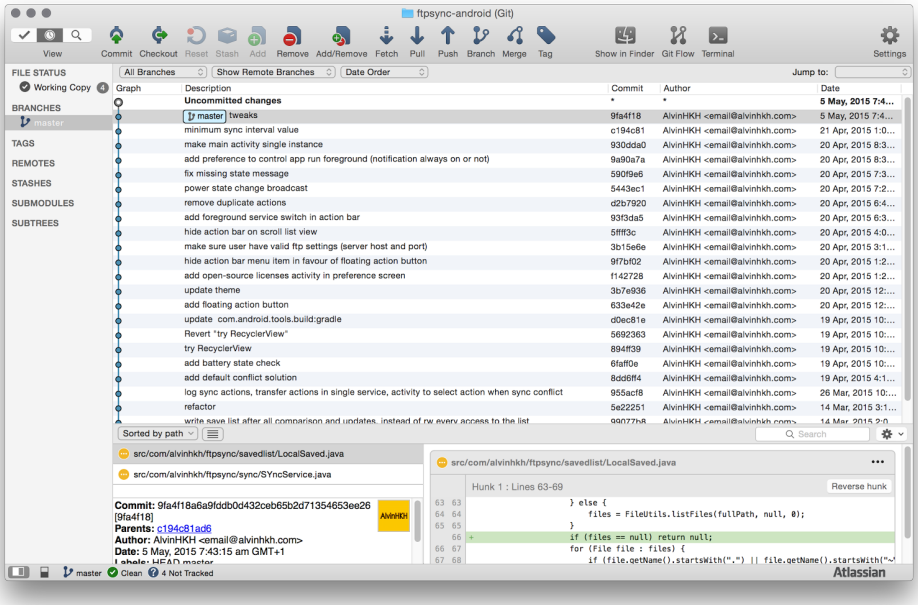
FileZilla FTP

During the development of file transfers and other ftps connections, I used FileZilla to check the directories and files on the server. I chose to use FileZilla as it is already install on my laptop and linked to my ftps server which use for testing during the project. This software offers a command-line console output, therefore I can check whether the ftps server is working properly. More importantly, I can check files synchronised correctly or not, and to make sure remote saved list is in correct format.

Git

Git was used to track changes made throughout the implementation stage. Git very useful to this project especially on the Android development. As I had no experience on developing Android applications, I tried multiple things like using different layout structure, i.e. ListView vs RecycleView. Experimenting with new methods do not always get optimum results; Git enable the ability to revert changes made to previous working state.

This also make backup and versioning a simple process, as it allow me to store changes at various stages in an increment way without the needs to backup the whole project every time. With the help of platforms like Bitbucket and Github, I can store a list of changes and a full backup in the cloud. My work has fewer chances to be total lost, as they are saved in local hard drive and in the cloud.



Screenshot 23. Use of Git for version control.

5.2. Methodology

5.2.1. Development Model

Originally, I was going to use Waterfall software development model, which means I need to fully complete each phase and I cannot go back and make changes. Testing can only be done at the end of the project, I found this model does not suitable my development process. Therefore, I chose to adopt a hybrid Waterfall-Agile methodology, where I allow changes to be made after the initial planning. Testing at each function implemented which ensures that the bugs are caught and taken care of in the development cycle. [13]

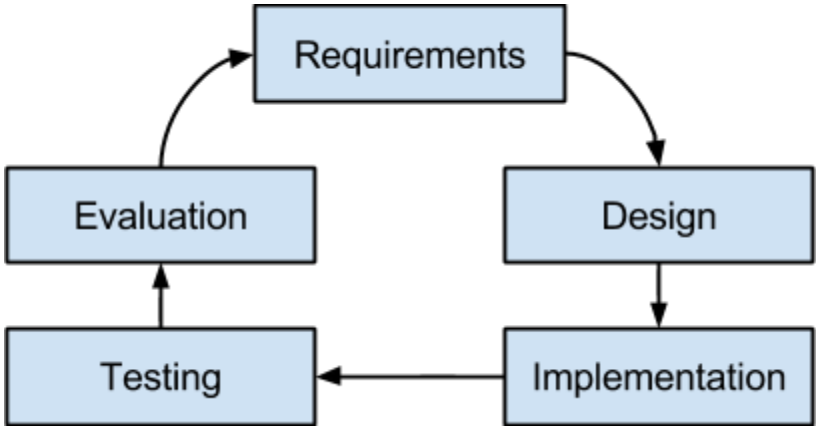


Diagram 9. Development Cycle

5.2.2. Android Development

For Android UI, I use Fragments and Tab view to show different sections, including, main section *(MainFragment)*, log section *(LogFragment)* and sync list section *(PathsFragment)*. All tabs fragments handle under MainActivity. The *ActionBar* contains a switch to enable auto sync service, and a button to navigate to preference screen.

I chose to use “NoNonsense-FilePicker” framework for folder picker when adding or editing sync directory. The framework includes local directory selection and support extend framework with other types. I implemented the ftps directory selection by extending the framework using *FtpsUtils* class, the class written for all kinds data connections to ftps server.

In addition, I develop a custom dialog box for editing path *(PathEditActivity)* and select file when sync conflict occurs. Dialog boxes show as a pop-up activity will not cover the

whole screen. For further enhancement, the file selection dialog can show as expanded notifications in order to avoid distributing users' activity.

Android provides a powerful xml driven framework to manage user preferences, that allows me to easily create preference screens using PreferenceFragment. However, the framework has limited default type of preferences. I added the *NumbersPickerPreference* for sync battery option *(SyncBatteryPreference)* and sync interval picker *(SyncIntervalPreference)*.

Receivers are part of broadcast system, some receiver tags are broadcast by system activity and some receiver tags are broadcast by self-implement classes. The *SyncReceiver* enable to set and trigger schedule synchronization automatically in a given time frame, when receive *com.alvinhkh.ftpsync.action.SYNC_CLOCK* tag.

In addition, I used several receivers to get system states, *AutoStartReceiver* to get *android.intent.action.BOOT_COMPLETED* signal that notify when device boot up completed, *NetworkStateReceiver* to get *android.net.conn.CONNECTIVITY_CHANGE* signal when network connection state changes, *PowerReceiver* to get *android.intent.action.ACTION_POWER_CONNECTED*, *ACTION_POWER_DISCONNECTED*, *ACTION_BATTERY_LOW*, *ACTION_BATTERY_OKAY*, in order to check battery status.

5.2.3. Python (Mac OS X) Development

For the Mac version, the whole application has a similar flow to Android application. For example, the use of an object to contain file transfer details, and the use of save list class to retrieve directory list. The main differences between Android version is the UI, except the programming language. For the main UIs, I implemented them using Tkinter module. Each section has its own frame, when user navigate to that section, the frame will bring to the front of the screen. The WindowFrame is the container of all frames and main ui, it handles ViewLogFrame, ViewSelectionFrame and UpdateSelectionFrame. These three frames represent view logs, view sync list and add/edit sync directory. The Preference screen is a separate window, each time the user hit save button, the application will check the ftps configuration and try to connect to the server. Shelve module is used to record logs and store sync directory. This structure enable expansion of frames; I can easily add new frame to the array in WindowFrame.

The whole application is navigated using menu in status bar tray. This menu bar used rumps module to implement. The menu includes links to Main Window, Preference Screen, button to sync manually, and turn on or off schedule sync. I did not implement the dialog window for user to select conflict file like the Android application due to time limit, but there is a setting to default choose local or remote file when conflict happens. Using the status bar is the most compact solution, as the application is designed to run in background with occasional access to log screen and set sync directory.

5.2.4. Different Approaches

The normal FTP protocol which does not secure and has many known vulnerabilities, like Brute force attacks, Spoof attacks and Username protection. There are several file transfer protocol with different security method, for example, FTPS, SFTP and FTP over SSH. In this project, I chose to focus on one protocol with secure data connection. Hence, I chose FTPS protocol, as FTPS is an extension to the FTP standard and it is the required protocol from project description.

Number of cloud solutions, such as Dropbox, Google Drive and iCloud Drive, are using a fixed parent directory and choose which sub-directories to be synchronise. I tend to think this restriction will limit the use cases but lower the chances of running into problems. This limitation may not major affect desktop users, however, this may be a big problem to mobile devices. As mobile devices usually have a small storage size comparing to desktops or laptops; in addition, some directories may not able to change under the application defined folder, for example, DCIM keeps captured camera photos, most of Android devices cannot change the path to other folders. If I adopt single fixed sync directory, these users will not able to synchronise this folder containing all photos taken. Therefore, I chose to adopt multiple sync directories with different parent folders.

To take care mobile users with limited network plan, the Android application applied the ability to sync only in WiFi condition. In addition, for phone with small battery, the application can limit synchronization while device is not charging.

Coding Approach

There are several ways to handle UI layer and background actions. Developments on Android using Java requires tasks separate from the main ui thread, especially operations requires network connection. This approach makes sure rendering ui will not block or wait forever until certain data thread loaded. FPS represents the responsiveness of the ui on devices. Mixing operation and ui thread will cause significant FPS drop which lead to low responsive between user touch and screen updates.

In addition, I implemented a FTPS connection class, *FTPUtils*, to operate all kinds of data transfer and request using *org.apache.commons.net.ftp* Java library. The class contains methods to download or upload single file or multiple files in a directory, remove files on remote server and request directory list… Besides, all file and transfer details like local path, remote path and sync status, are transmitted using a class object, *FileTransferHelper*. The class is *Parcelable* object, therefore, data can pass from background services to main activity and other services via broadcast. An alternative way to implement such functions is implement in a single class file, however, separating different modules help organise different part of the program.

As the ftps protocol does not support request server time, in order to check the time difference between local client and remote server. First set the ftps connection in local time zone, then create and upload a temporary file from local to server. Finally, compare

its last modification date with current time on the client device. This approach can give extra insurance for setting timestamps on files. This is useful for time-based synchronization, however, after consideration, I adopt the version-based synchronization, this method is less important.

A good code quality and practice led to a successful, readable and expendable outcome. The tools I use for developing two applications provide guides to follow recommend variable and method naming guidelines. Whenever the name of variables and methods, the IDEs will highlight errors.

I tried to keep different functions in a separate classes to ease maintenance and development. If everything are put in one component, it becomes entangled and causes problems with UI crashes on Android. Separating different modules help to debug the application and ease future enhancement,

In Android, to save records of sync directories and logs, I need a database. I chose to use SQLite with own implementation of content provider. To provide access to the database using Java classes. This gives me a more flexible way to deal with database data and display items on ListView element.

There are different ways to perform jobs in background, asynctask, intentservice and service. I initially used asynctask to perform sync operation, but later I found out if I use asynctask, I will encounter problems when extending to background sync, due to asynctask need to trigger from main thread. While IntentService provides a working queue for long tasks, file transfer can be a long task if file size is huge. The following table is the comparison between different options. [14]

	Service	Thread	IntentService	AsyncTask
When to use ?	Task with no UI, but shouldn't be too long. Use threads within service for long tasks.	- Long task in general. - For tasks in parallel use Multiple threads (traditional mechanisms)	- Long task usually with no communication to main thread. - If communication is required, can use main thread handler or broadcast intents - When callbacks are needed (Intent triggered tasks).	- Small task having to communicate with main thread. - For tasks in parallel use multiple instances OR Executor
Trigger	Call to method onStartService()	Thread start() method	Intent	Call to method execute()
Triggered From (thread)	Any thread	Any Thread	Main Thread (Intent is received on main thread and then worker thread is spawned)	Main Thread
Runs On (thread)	Main Thread	Its own thread	Separate worker thread	Worker thread. However, Main thread methods may be invoked in between to publish progress.
Limitations / Drawbacks	May block main thread	- Manual thread management - Code may become difficult to read	- Cannot run tasks in parallel. - Multiple intents are queued on the same worker thread.	- one instance can only be executed once (hence cannot run in a loop) - Must be created and executed from the Main thread

Table 1. Comparison between Service, Thread, IntentService and AsyncTask.

5.3. Encounter Problems and Solutions

During the implementation stage, I encounter few problems when developing different functions and features. Below is a list of problems and corresponding solutions.

Problem	**Solution**
When there is an ongoing progress dialog box, UI may crash if rotating the screen in Android. In addition, the framework I use for folder picker when using *AyncTask* to get FTPS directory list, the application may crash.	A workaround is to disable screen orientation detection while displaying the dialog box or implement Handler to show messages. At the end, I chose to show progress bar on UI and notification bar, instead of popup progress dialog box.
When clicking the notification, user can return to the application. However, it creates a new activity instead of opening existing window. Widgets may not showing messages from broadcaster correctly.	use `android:launchMode="singleInstance"` to avoid crash on UI elements and widgets not showing correctly when messages get by broadcaster.
ftps protocol do not have command to request server time, time might be wrong and not synchronise between client and server.	Let user to select which files remain is one of the possible solution for such conflict when file synchronise.
Android has a bug on *setLastModified* method call. [15]	Standard method call is in place, if bug is solved or particular device support *setLastModified* then it can run without workaround. Otherwise, the application will call su command to set time, which needs root access.
Limitation on time for testing on various version of android and make code compatible with older devices. I only own a mobile device with the latest version of android at this period. As previous	As I own a Nexus 6 (Android 5.0 Lollipop), I target the application to Android API level 21, version 5.0 Lollipop.

problem with set last modification date, I cannot simply run the application on emulator, hence, no devices for testing.	
Standard types of Preference is not suitable for picking multiple numbers.	Instead of making it a text field to enter numbers. I created custom preference, *NumbersPickerPreference*
Tkinter module seems do not have the ability to create Mac OS X status bar tray icon Window	Use rumps module to handle the status bar tray app window. The module provides an interface with few method calls to handle menu icon and items click actions
Using ftplib module to connect ftps server under TLS. *prot_p()* method call is needed in order to secure all the file data connection, not only the user and password is encrypted. But some ftps server may fail to work with this function call. [16]	Make a workaround to the issue by setting this function call optional in preference.

5.4. Limitations

There are few limitations of the current implementation of both Android and Mac OS X application. The Android application is target version Android 5.0 and above, this issue can be solve in short term by giving more time to extend and test the compatibility.

Besides, there is a bug in Android [15] that block the use of setLastModified in java.io.File class to change timestamps of a file. A workaround implemented by using su command to set timestamps, however, this means the application require root access which general users might not have it. According to comments in the issue ticket some vendor implemented some workarounds in system level [17], which make setLastModified method usable. If particular devices support or Android solve this bug, the application can run without root access, as code to use standard method to modify timestamp already in place.

In addition, in version 4.4.2, Android has limited applications access external sd card [18]; this issue should be solved in Android 5.0 with new API and permission model, but I do not have a device with external sd card support. If this bug still exist in 5.0 or when the application extend to support Android 4.4, this will be a limitation due to user would be able to synchronise directories in external storage.

Regards to Mac OS X application, the script written in Python using ftplib module to connect to ftps server. *prot_p()* method is to secure data connection, however, some server may not support implicit connection [19]. As mention in above section, I made this function optional in preference to make the system usable.

On both applications, the system is design to work with FTPS protocol only. With a little extra effort they can support ftp protocol, which carry data in lower security but with a higher transfer data speed.

6. Testing and Evaluation

6.1. Test Cases

There are few preconditions before the test, user must have their own ftps server set up, and they must have access right to the server. Due to the fact that Android on some devices has a bug blocking application to set last modification date of a file using standard Java function call, root access is needed in order to workaround the issue. I tested each functional requirements set at the beginning of the project, and the full results are listed in Appendix D.

Testing Environments

- *Mac OS X platform:*
    - *Macbook Pro (Retina, 13-inch, Mid 2014), OS X 10.10.3 Yosemite*
- *Android platform:*
    - *Nexus 6, Android 5.1.1*
- *FTPS Server:*
    - *Own FTPS Server located in Hong Kong*

The most important test for this project is to ensure sync method I implemented work as intended. This operation is a combination of multiple actions and test case. First, the application needs to retrieve list of local directories; then, perform comparison between current and previous; repeat the same with remote directories; after that, compare the latest local list and remote list; during the comparison, files will be identify to do certain actions; files need to update to server will upload from local, files updated on server will download to local, files removed will mark as deleted on the list; finally, save the latest list and notify users action completed. So far, the application works in ideal situation with controlled environment.

Most of the tests against functional requirements pass the test; however, there are things that indicated improvements are needed. First, when network connection is unavailable, Android application will show connection unavailable, and Mac OS X application will stop working without any notice. Both applications need to implement a better solution to let user know their network connection affects the application to synchronise data. Secondly, if directory disappear after last sync and directory exist on either local or remote side, synchronization run normally. Otherwise, directory disappear may cause data loss.

On the other hand, there are some situations where cases could not be tested. For examples, situations where there are lots of clients that the server and the saved directory list are not workable, devices with other versions of OS and ftps server without user name and password.

6.2. User Evaluation

With the user evaluation, I adopted the think-aloud evaluation. Basically, when the tester doing each evaluation tasks, they speak out their thoughts, such as, things they are try to do and things they read; what make them take series of actions and questions that arise in their mind; and most importantly things they find confusing, so I can improve the application.

To maximise the outcome of this test, I follow the protocol for six stages to conduct a user test. First, I developed the test plan by using use cases and the specification developed.

Second, as one of the pros of think aloud evaluation is to do with non-expert; I invited my house mates as participants of this evaluation.

Third, I prepare a video clip as an example of think aloud user test in order to give a brief idea to the participants what they need to do during the test.

Fourth, I did a pilot test once for myself to check everything run smoothly and work well.

Fifth, while conducting the real test, there is a facilitator that keep reminding users to think aloud by asking them questions, for example, things they are thinking right now, the reason they are doing such actions or responses, etc. During the test, I act as an observer and being quiet to take notes about things happened especially those critical incidents that affect user experiences and satisfaction.

Finally, I did the analysis and report for the testing process and the results are list below. I decided to use Nielsen's 10 usability heuristics to show each usability problem user encountered; this enhanced our evaluation process of testing (Nielsen Norman Group,1995).

The tables below define the severity and ease of fix rating systems applied. Severity ranks are based on those defined by Jakob Nielsen (Severity ratings for usability problems) [20].

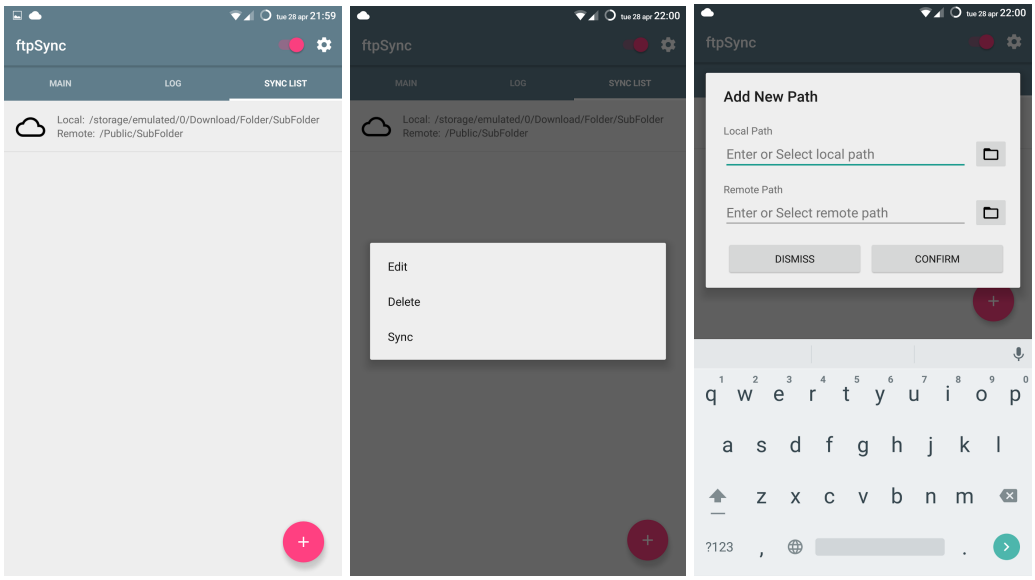
Level of Severity Ratings

Rating	Description
0	Violates a heuristic but doesn't seem to be a usability problem.
1	Unimportant usability problem: does not occur often and can be easily overcome by the user. Fixing should be given very low priority.
2	Minor usability problem: this problem occurs more frequently and is more difficult for the user to overcome. Fixing should be given low priority
3	Major usability problem: this problem occurs very often and users are unable to fix the problem. Very important to fix.
4	Usability catastrophe: this problem impairs use of the product by users cannot be overcome. This should be given the highest priority to fix.

Ease of Fixing the Problem Ratings

Rating	Description
0	Problem is very easy to fix
1	Problem is easy to fix
2	Problem would require more effort to fix, the problem may involve more than one aspects of the interface
3	Problem would be problematic to fix, it may involve a number of sections of the interface

Task - View and Add Sync Path using Android application

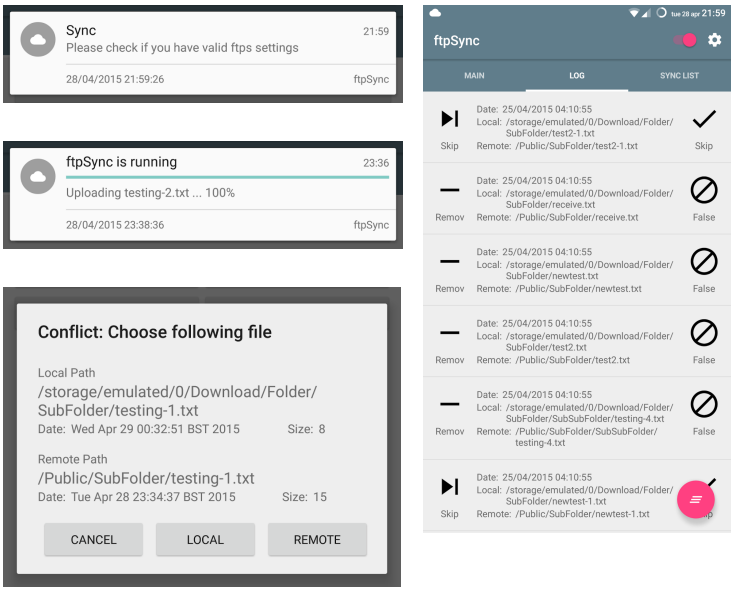


Usability Principle	**Complies (Y/N/NA)**	**Note**
Visibility	Y	User can identify that they are adding new sync path to database or viewing list of sync path, and understand what actions are possible.
Speak the user language	Y	Users should recognise required input and message from experience. No technical terms are used.
User Control and freedom	Y/N	Users are able to cancel the operation by clicking dismiss button and quit the current activity. Delete record do not have any notice, and not be able to undo.
Consistency	Y	No technical words are used. Icons come from Google Material Design Guideline, users should feel natural in Android platform. Add button on view list screen adopt floating action button introduced by Google.
Error prevention	Y/N	When either local or remote path is empty, new entry will not save and notify user. In view list screen, user unable to tell whether the sync directory exist and available or not.

Recognition vs. recall	Y	Appropriate controls are provided, for example, button to “Submit” and “Dismiss”, folder icon to select directory using folder picker.
Flexibility	NA	
Minimalist design	Y	Flat and clean design. Some graphics are used to represent meaning of some text.
Recognise, diagnose and recover from error	N	Accidentally delete entry without notice.
Help	Y	A message shown when there are no directories selected.

Usability Problems outlined in feedback sessions	**Usability Guidelines these problems violate** [21]	**Severity Rating**	**Ease of Fixing Problem**	**Recommendation**
Users accidentally delete records and not able to undo	User control and freedom: There is no indication on the screen to notice users that record will be deleted when pressing “Delete”. Recognise, diagnose and recover from error: There is no way to undo this delete decision.	4	1	Add a confirmation pop-up dialog box.
User unable to tell whether the sync directory exist and available or not.	Error prevention: In view list screen, there is no indication that selected directories are available and have right to access and write files.	3	2	Check directory status when refreshing the list, and update the cloud icon on the left of each row to indicate the availability.

Task - Manually Sync using Android application



Usability Principle	Complies (Y/N/NA)	Note
Visibility	Y/N	User can see the progress of synchronisation through notification and progress bar. First-time users may not have a clear instruction to set up all necessary settings.
Speak the user language	Y	Users should recognise sync button to initiate the action.
User Control and freedom	N/Y	User unable to cancel or stop ongoing sync process. User able to select options to resolve conflict files.
Consistency	Y	No technical words are used. Icons come from Google Material Design Guideline, users should feel natural in Android platform.
Error prevention	Y/N	If user do not have a valid ftps settings to continue the sync process, or users are not in valid condition like battery is low, proper messages will be shown.

		No message shown when start synchronisation with empty list of sync directory.
Recognition vs. recall	Y	Same notification will show up each time. Same sync button and progress bar on main screen.
Flexibility	Y	User able to set default action when file conflict occurs.
Minimalist design	Y/N	Notification provide relevant and important information regards to sync progress. Resolve conflict dialog box contain necessary information, like file size and date, to let user choose suitable file. Log Screen is too detail for novice users.
Recognise, diagnose and recover from error	Y	All error messages like require valid ftps settings are expressed in plain language and precisely indicate the problem in notification.
Help	N	There is no documentation or user guide especially for first-time users to begin with.

Usability Problems outlined in feedback sessions	**Usability Guidelines these problems violate []**	**Severity Rating**	**Ease of Fixing Problem**	**Recommendation** *(More details in below Future Enhancement section)*
No ways to cancel operation when users started the sync process	User control and freedom: Once sync process started, there is no way to cancel or stop the operation. There is no “emergency exit” for users.	4	3	Implements the emergency exit in *TransferService* to detect whether cancel operation flagged.
No message shown when start synchronisati	Error prevention: There are no messages or notifications to notify users that the list is	3	1	Check Sync List every time synchronization start.

on with empty list of sync directory.	empty, except a message on the list screen.			Ask user to add sync directory when user launch the application first-time after setting the ftps configuration.
Log is too details, but no general overview of each sync action	Minimalist design: Log Screen contains too much information for novice users. Visibility: There is no overview of each sync, i.e. sync status of sync directory as a whole, not individual files inside the directory.	2	3	Add a filter control in log screen to select date range, transfer actions, or sync status. Add new type of log to record overview status of sync directory.
First-time Instructions	Visibility of system status: There is no instruction at any time. When ftps server settings are empty, users are not able to leave preference screen. Help and documentation: It is straightforward to use, once users find out how at first. But there is no user guide or flow for first-time users.	1	2	Add a user guide for users run the application for the first time. Help users to setup necessary settings.

7. Conclusion

The aim of this project is to build a DIY cloud solution, based on an FTPS server controlled by the user. Overall I believe the project to be a success as I achieved most of the requirements set at the beginning of the project. I developed an application for Android devices using Java and one for Mac OS X platform using Python. The application able to synchronise selected directories with ftps server.

Even though there are many uncertain in my initial plan, I ultimate decided things I need to work on. Although the project has been a challenge, it has also been entertained to learn about new aspects. As a result of this project, I learnt a lot throughout the process; I gained hand-on experiences on programming in Android platform, a wider knowledge of methods Java provided, the use of tools like Android Studio and PyCharm, and problems and solutions to remote synchronization issues.

The outcome of this project effectual done various things. The system is implemented on Android and Mac OS X platform; it able to use ftps protocol to synchronize directories between local client and remote server automatically in a given time frame. Most of the requirements are implemented; the results shown in testing section above. In addition, some problems found during user evaluation; some fixes has been made according to the recommendation and some are listed in future enhancement.

Definitely there are numerous areas that can be improved within both applications. For examples, current execution do not support multiple devices on same platform; it depends on a single file to retrieve the directory list; and it only sync folders but not single file; others are outlined in future work section. Both applications are still supposed to be a prototype of this project idea, they are not ready for prime time.

8. Future Enhancement

Throughout the project I came up with some additional ideas. I have implemented some of them and indicated above, for example, a log screen. However, there are many more ideas that were not achievable in the time frame given. As I try to stick with the work plan set at the beginning of the project; if this project was run two semesters instead of one, I would do much more than currently achieved.

There are numerous further enhancement can improve and add to the applications to make the system better. Some of these ideas already mentioned in different sections in the report. Below is the list of opportunities that can enhance the application.

Improvements on User Experience

- Cancel Sync Action: Let user to cancel the operation at any time at any stage of the synchronization process.
- Log Filter: Current implementation of log record and show all transfer actions including those file skipped. In addition, logs are not recording synchronise directory set in general view. i.e. Set sync “Folder_A” and log show transfer actions for all files inside “Folder_A” but not generally show this folder is synchronising or finished.
- User Guide: A user guide for first-time user to help user to set up necessary settings like ftps server settings and enable scheduled sync. Current implementation for Android app is user unable to navigate to other activity when they have invalid ftps settings. They can either continue edit preferences or exit the application.
- Make it compatible with older version of Android.

Improvements on Sync Algorithm and Flow

- Sync individual file: As a workaround, user can now make a folder and place one file in it. The underlying code for synchronising single file is almost complete, but there is one issue that block this feature listed as implemented. The problem is related to handle record of saved directory list for single file.
- Saved List in Database: Hence, in order to improve both showing sync directories status and handling save list entries for single file. Instead of using .ftpsync JSON file to save the directory list and each file version number, the application can add a database to store each sync file entries. This will enhance and solve followings:

    - Improve comparison between files from difference time by having a longer list of history of files, currently only the last synchronise list.
    - Save latest list when each transfer action complete, rather than save it once at the end of synchronisation.
    - Avoid save list .ftpsync file deleted by user accidentally using other file manager apps.
- Backup Files: Current implementation of replacing existing files is, first, replacing the existing file with a different name, then upload the new file, remove the old file. A new approach can be done to lower the chances of loss of data during sync. Delete files move to a folder act as recycle bin, instead of delete it directly. Store both files when sync conflicts with the client names, therefore, if anything happens before user select which file is the latest one, both file will not loss and exist in both local client and remote server.

Improvements on User Interface

- The user interface of Android application looks good at this moment, however certain screens do not scale properly on larger devices. Despite these screen works on tablets but the size of buttons and layout configurations need to improve for a better user experiences.
- The user interface of Mac OS application does not native to system enough, except the status bar menu which uses *rumps* module, there is a need to choose another python module to build the user interface elements instead of *Tkinter*. This needs more research on different python modules that draw window interface on Mac OS X platform, therefore, I have not rewritten those windows on the Mac app.
- As mention above, this application can be improved by adding a user interface clearly show file status of each sync file, but not the log of transfer action. i.e. visualise the saved directory list with version numbers of each files.
- Folder Picker: Implementing own folder picker with better user interface that matches material design which let to better user experience. In addition, reimplement the ftps folder picker with a faster loading speed on requesting ftps server file list, for example, cache the parent and loaded directory lists so request to server is not needed when navigating previous loaded directories.

More enhance features

- Multiple FTPS server account: Instead of connecting one ftps server, the application can add multiple ftps server, therefore users can sync across different servers. Moreover, the application can act as intermediary between two ftps server.
- Server Mode: Run application with built-in ftps server, along with sync client, to make it easier for general users to adopt cross-devices file synchronization.
- More platform support: Implement the system on other platforms, for example, Windows. As Python and Java are cross-platform programming language, they should be able to run on other platforms with some addition framework to build native user interface.

9. Reflection

Throughout this assignment, I have couples of new challenges and gain some new skills. This project provided me with an exciting opportunity to apply what I have learnt in project planning, design and some HCI; in addition, throughout this project, I gained some new experiences and skills on programming and time management.

This is my first experience with Android development; I was not sure which framework and approach should use to begin with. Design well before implementation is important to led to successful outcome. Initially, I tried to work on timestamps-based synchronisation; as mentioned in the report, timestamps-based sync has its limitation, in addition to the bug on android that causes unable to modify timestamps, I found out current solution to use version vector is better option. More research and planning before implementation assist me to know my skill limitation, and limitation of different programming languages, platforms, modules and frameworks.

Using one programming language may be faster to implement; it depends on project I work on and things that need to develop. In this project, I do not feel comfortable enough to use Java on both Android and Mac OS X, as it would be two new platforms for me to develop. I have had experience on developing Python application on desktop and writing code in Java during time at university, but I had no experiences on Android platform before this project. In the future, I should take more factors to consider and do more research when implementing a system on two different platforms using two different languages at a given time frame for a single project.

Moreover, I learnt the usefulness of keep a record of decision on different approaches I took to complete each tasks; these notes recorded help me to show my critical thinking on vital aspects in this report. Besides, I made use of some HCI skills developed in second year module, to take care user experiences and conduct the user evaluation in testing stage.

Regards to time management, I tend to think the biweekly work schedule planner is well operated. As the planning for each tasks are high-levels, such planning gives a suitable time span to implement and test those planned features, which led to a successful completion of the project. However, I found the time plan should put more time on implementation and it should start earlier. A more compact schedule may help to leave more time at the end, to test and improve the application, especially when working with a new platform. I feel myself lack of time to perform a more comprehensive testing on both applications. Alternatively, I would plan everything one or two weeks earlier, as I was able to maintain to stick with my initial work plan at the beginning of the project; This way I would be able to leave more time to document all my steps and approaches to solutions, and testing. In this project, I learnt to be more organized, to establish a working plan and to follow this plan strictly; furthermore, I learnt to prioritise importance of tasks that need to conduct and take consideration that risks might encounter.

To conclude, I build up programming experiences on Android platform with Java, and Mac OS X with Python. I took a lesson on well-design and better work plan make better execution with less time wasted. I have also learnt the importance of time planning and management. Additionally, I used knowledge from previous projects to help to build a user-friendly product with HCI to take care user experiences, and other report writing skills. I look forward to taking these skills further and developing them as I move through my professional career.

References

[1] Lucas Mearian, Computerworld, 2013. *No, your data isn't secure in the cloud* [Online]. Available at: http://www.computerworld.com/article/2483552/cloud-security/no--your-data-isn-t-secure-in-the-cloud.html [Accessed: 30 January 2015].

[2] SparkleShare, 2015. *WWW* [Online]. Available at: http://sparkleshare.org/ [Accessed: 20 April 2015]

[3] OwnCloud, 2015. *WWW* [Online]. Available at: https://owncloud.org/features/ [Accessed: 20 April 2015]

[4] Syncany, 2015. *WWW* [Online]. Available at: https://www.syncany.org/ [Accessed: 20 April 2015]

[5] Syncany, 2015. *What is Syncany?* [Online]. Available at: http://syncany.readthedocs.org/en/latest/what_is_syncany.html [Accessed: 20 April 2015]

[6] prikryl, WinSCP, 2013. *Synchronizing* [Online]. Available at: http://winscp.net/eng/docs/task_synchronize [Accessed: 20 April 2015]

[7] Bryan, 2010. *Why Vector Clocks are Easy* [Online]. Available at: http://basho.com/why-vector-clocks-are-easy/ [Accessed: 30 April 2015]

[8] makovkastar, 2015. *FloatingActionButton* [Online]. Available at: https://github.com/makovkastar/FloatingActionButton [Accessed: 10 March 2015]

[9] spacecowboy, 2014. *NoNonsense-FilePicker* [Online]. Available at: https://github.com/spacecowboy/NoNonsense-FilePicker [Accessed: 10 March 2015]

[10] Google, 2015. *Material Design icons by Google* [Online]. Available at: http://google.github.io/material-design-icons/ [Accessed: 10 March 2015]

[11] jaredks, 2015. *Ridiculously Uncomplicated Mac os x Python Statusbar apps* [Online]. Available at: https://github.com/jaredks/rumps [Accessed: 10 March 2015]

[12] py2app, 2015. *WWW* [Online]. Available at: https://pythonhosted.org/py2app/ [Accessed: 10 March 2015]

[13] Base36, 2012. *Agile & Waterfall Methodologies – A Side-By-Side Comparison.* Available at: http://www.base36.com/2012/12/agile-waterfall-methodologies-a-side-by-side-comparison/[Accessed: 1 March 2015]

[14] Tejas Lagvankar, 2011. *Android Thread Constructs(Part 4): Comparisons* [Online]. Available at: http://techtej.blogspot.co.uk/2011/03/android-thread-constructspart-4.html [Accessed: 10 April 2015]

[15] *setLastModified() always fails on Xoom unless running as root, Android Open Source Project - Issue Tracker* [Online]. Available at: https://code.google.com/p/android/issues/detail?id=18624 [Accessed: 10 April 2015]

[16] Ye.Wang, 2013. *Error when connecting to FTPS servers not supporting SSL session resuming* [Online]. Available at: http://bugs.python.org/issue19500 [Accessed: 10 April 2015]

[17] Workaround to setLastModified() [Online]. Available at: https://code.google.com/p/android/issues/detail?id=18624#c50 [Accessed: 10 April 2015]

[18] *MicroSD access for applications on Google Play edition devices, Android Open Source Project - Issue Tracker* [Online]. Available at: https://code.google.com/p/android/issues/detail?id=63879 [Accessed: 10 April 2015]

[19] Jeremy.Brock, 2012. *FTP_TLS in ftplib not supporting prot_p storlines in FTP7.5* [Online]. Available at: http://bugs.python.org/issue16318 [Accessed: 10 April 2015]

[20] Jakob Nielsen, 1995. Severity Ratings for Usability Problems [Online]. Available at: http://www.nngroup.com/articles/how-to-rate-the-severity-of-usability-problems/ [Accessed: 30 April 2015]

[21] Nielsen Norman Group. 1995. *10 Usability Heuristics for User Interface Design* [Online]. Available at: http://www.nngroup.com/articles/ten-usability-heuristics/ [Accessed: 1 May 2015]

Appendices