Software Engineering Project
Project Group 4

Technical Documentation

The SyncBox Project Team:

Arpit Agarwal - 11010107
B.N.Karthik – 11010114
J. Surendranath Reddy – 11010123
Prudhvi Raj Chowhan – 11010131
Manikanta Reddy – 11010148
Rahul R. Huilgol - 11010156
Sachin Aglave – 11010160
Sparsh Kumar Sinha – 11010166
Vishal Anand – 11010170
Shobhit Chaurasia – 11010179
B Sri Harsha – 11010110
Harsha S. Tirmala – 11010120
K. Dinikar Reddy – 11010127
Neha Damadya – 11010143
Rachit Kumar – 11010154
Rakshita Jain – 11010157
Shivam Kumar – 11010164
Venkat Abhinav – 11010169
Shyamal Kejriwal – 11010174

Preface

The document herein was produced by the ***SyncBox Team***, a group of highly enthusiast Computer Science undergraduates of the Indian Institute of Technology Guwahati. This document is intended to provide a technical over-view of this project, SyncBox, a web-based file-sharing portal with the provision of auto-synchronization of files as the user is on the move.

This project was developed mainly in the timespan of ONE month, during March 15, 2013-April 17, 2013. Developed under the guidance of *Prof. (Dr.)* ***Pradeep Kumar Das***, this group could visualize the importance of working as synchronized chunks of software development units and could complete this project, while enjoying the challenges that crept up. There are no restrictions on the reproduction, distribution, translation or use of this document. However, incorporation of this document, in part or in whole, into any other document does not convey or represent an endorsement of any kind by the SyncBox Team.

The SyncBox Team
2nd Year,
Department of Computer Science and Technology,
Indian Institute of Technology Guwahati
Guwahati – 781039,
India

Date – 17th April, 2013

Introduction

This is an introduction to the technical descriptions of the Project on Web-Based-File-Hosting-Server. This Project has been developed to construct a web-portal to come up with these functionalities:

- User Authentication
- File upload
- Directory Upload
- Directory Synchronization
- Complete web based file browser with complete details of files.
- Allow file sharing: specific/ group/ public.
- Allow to open basic file types doc, docx, xls, xlsx, ppsx, images with slide show, text and multimedia in browser.
- Provide file encryption
- File compression
- User to user and admin - user communication
- Use proper database for storing all relevant data.

The ensuing literature dictates some of the technical aspects of this open-source software that we have developed for the user.

1.0 Website

1.1 Framework Used

We have used the Django[1] Framework to build up the website pertaining to this project. For the uninitiated, Django is a free and open source web application framework, written in Python, which follows the model-view-controller (MVC) architectural pattern. Since this project uses intensive sqlite3 database use, Django facilitates us with the ease of access in such scenarios.

[1] www.djangoproject.com/

For the basic aesthetic look of the website, we have used the Twitter Bootstrap[2] CSS package to avoid re-defining the classes of the CSS classes for the different subsections of the web-site.

1.2 Purpose

Although there exists a separate programme to run in case one has to synchronize the files, still we have come up with the website in case one would like to view it from the browser, while browsing for some other stuff. Although it has an added feature of signing up, which is not present in the executable version.

1.3 Scope

We can allow for the upload of files directly to the user's account, and even the deletion of the file(s). When this particular user connects to the server from the personal computer, these files would be synchronized to that particular computer and thus the user who might or might not have the PC available to himself/herself, would still be able to properly store the files as per the needs of the hour.

2.0 *Installation and Authentication* on First-Run

2.1 Installation of Dependencies

To use synchronizing feature, a few packages are required:
(i). Unison[3]
(ii). iNotify[4] Tools
(iii). Open SSH Client[5]

These need to be installed at the first execution of the software. So, we are providing an app to ensure these dependencies are installed after the user provides us with his/her system password. Along with these dependencies, an SSH key is also generated and is stored in the appropriate text file, which needs to be sent to the

[2] twitter.github.io/bootstrap/
[3] *www.**unison**.org.uk*
[4] en.wikipedia.org/wiki/Inotify
[5] en.wikipedia.org/wiki/Secure_Shell

computer where the server script is running so as to establish a connection.
If the packages fail to install for some reasons, then the user has to install them manually using the instructions provided in app.

3.0 FILE SYNCING APPLICATION

3.1 Framework Used

PyQt-4 framework has been used to prepare the GUI for the application.

3.2 Purpose

This application serves the following essential functions:

- User login username/password
- Proxy settings for the file syncing
- Proxy username / password
- File path/folder select to sync
- Start Syncing
- Error messages
- Auto run on startup

3.3 FUNCTIONS

Username can be entered as a string input to the following function :

```
self.lineEdit = QtGui.QLineEdit(self.frame) #to take username
      self.lineEdit.setGeometry(QtCore.QRect(100, 26, 201, 31))
      self.lineEdit.setObjectName(_fromUtf8("lineEdit"))
```

Password can be entered as a string input to the following function :

```
self.lineEdit_2 = QtGui.QLineEdit(self.frame)#to take password
      self.lineEdit_2.setGeometry(QtCore.QRect(100, 70, 201, 31))
      self.lineEdit_2.setEchoMode(QtGui.QLineEdit.Password)
      self.lineEdit_2.setObjectName(_fromUtf8("lineEdit_2"))
```

proxy username / password taken as strings :

```
self.lineEdit_3 = QtGui.QLineEdit(self.frame_2) #to take proxy/
username
      self.lineEdit_3.setGeometry(QtCore.QRect(100, 26, 201, 31))
      self.lineEdit_3.setObjectName(_fromUtf8("lineEdit_3"))
      self.lineEdit_4 = QtGui.QLineEdit(self.frame_2) #to take proxy
password
      self.lineEdit_4.setGeometry(QtCore.QRect(100, 70, 201, 31))
      self.lineEdit_4.setEchoMode(QtGui.QLineEdit.Password)
```

The following code snippet executes the login function :

```
def login(self):    #login function
                                  self.textEdit.setText("")
                                  try:
                                   f=open('./id_rsa.pub','r')
                                   z=f.read()
                                   z=z[:-1]
                                  except IOError:
                                   self.textEdit.setText("Install ssh server")

      x=str(self.lineEdit.text())
      y=str(self.lineEdit_2.text())
```

3.4 SCOPE

User can login via this application to the username/password given at the website. He can then add the paths of the file/folders that are to be added to his/her account on the file sharing server . Common errors by the users that have been accounted for include :

- If user’s account has been deactivated , upon trying to log in an error message crops up as follows "Account is disabled".
- If wrong username and/or password are used then upon pressing the login button this message is displayed "Incorrect query".

- If no file path is selected then the following error message is shown : "Please select a directory to sync" .
- If unison file synchronizer is not installed an error "Error:Install unison for file sharing" .

3.5 UNISON

Unison is a file-synchronization tool for Unix and Windows. It allows two replicas of a collection of files and directories to be stored on different hosts (or different disks on the same host), modified separately, and then brought up to date by propagating the changes in each replica to the other.
Unison works *across* platforms, allowing you to synchronize a Windows laptop with a Unix server, for example. Unlike simple mirroring or backup utilities, Unison can deal with updates to *both* replicas of a distributed directory structure. Updates that do not conflict are propagated automatically. Conflicting updates are detected and displayed. Unison is resilient to failure. It is careful to leave the replicas and its own private structures in a sensible state at all times, even in case of abnormal termination or communication failures.
Unison works between any pair of machines connected to the internet, communicating over either a direct socket link or tunneling over an encrypted ssh connection. It is careful with network bandwidth, and runs well over slow links such as PPP connections. Transfers of small updates to large files are optimized using a compression protocol similar to rsync. Unlike a distributed filesystem, Unison is a user-level program: there is no need to modify the kernel or to have superuser privileges on either host.

Unison has a clear and precise specification. .Unison is free; full source code is available under the GNU Public License.

3.6 Inotifywait

Inotifywait "waits" for changes to files using inotify which is a linux system call. It efficiently waits for changes to files using linux's inotify interface. It is suitable for waiting for changes to files from shell scripts .It can either exit when an event occurs or continually execute and output events as they occur.

WORKING:

- Once the user logs in with the given credentials, he is allowed to choose a folder which will be synced. These credentials are saved in .ssh folder so that the user doesn't have to enter them again and again.
- Once the directory has been chosen, the user can start syncing
- Once start sync is chosen, we run the script main.sh
- The preferences for syncing are stored in .unison folder in test.prf
- When syncing starts, main.sh runs. This starts syncing by running 'unison test'. Then we use inotifywait to watch for changes in the given folder. In case of changes to the folder we run unison test again.
- The unique point is we don't copy the whole folder being synchronized, we only send those files which have changed

SERVER SIDE WORKING

- The folder corresponding to the user on the server directory has been changed by the app on client side.
- Once this has been done what remains is to add the list of files changed to the database
- We do this by running a script addtodb.sh. This

4.0 PROCESS-FLOW

First install the dependencies

```
ex: sudo apt-get install unison

        openssh-client (on client)
        openssh-server (on server)
        unison (on client and server)
        inotify-tools (on client)
```

Now run the 'ssh-keygen' to generate the public and private keys of the client and stores them in the .ssh folder on the client side. When the user tries to login for the first time using the application then the keys

are passed along with the username/password through ssh and are added to the ‘./.ssh/authorized_keys .
Send public key:
Send the contents of './.ssh/id_rsa.pub' on client to server ; The server receives this file and appends it to './.ssh/authorized_keys'
then write the next four lines to the file './.unison/test.prf'

```
root = ssh://SERVERHOST:PORT/SyncBox/$username/
root = $user-directory
batch = true
auto = true
```

Normal Run:
On startup of computer run './main.sh' - put this script wherever you want to and run it accordingly. './main.sh' runs unison and watches the folder (mentioned on the login page of the application while logging in for changes in which case unison is run again and so on….

5.0 REFERENCES

- **Unison** :
  http://www.cis.upenn.edu/~bcpierce/unison/index.html
- **Inotifywait :**
  http://linux.die.net/man/1/inotifywait
- **Django :**
  https://docs.djangoproject.com/en/1.5/
- **Bootstrap :**
  http://twitter.github.io/bootstrap/
- **SSH :**
  http://www.openssh.org/