Mobile FTP Client: An Android Application

By: Eric Sobel

Computer Engineering Department

College of Engineering

California Polytechnic State University

June 3, 2015

Table of Contents

Abstract **3**
1 Introduction **4**
1.1 Problem Statement 4
1.2 Solution 4
2 Frontend **5**
2.1 Design 5
2.2 Establishing a Connection 6
2.3 Permissions 7
2.4 Directory listing 8
2.5 Transfer 9
3 UI **10**
3.1 Design 10
3.2 Main Login Screen 11
3.3 File Listing Screen 13
3.4 Download Screen 14
4 Server **16**
4.1 Configuration of Server 16
4.2 Configuration of Router 17
5 Testing **18**
5.1 FileZilla 18
5.2 Pure Java FTPS 18
5.3 Android with FTPS 19
6 Related Work **20**
6.1 AndFTP 20
6.2 FileZilla 21
6.3 Nas4Free 22
7 Conclusion **23**
8 References **24**
9 Analysis of Senior Project Design **25**
10 Appendix **28**

Abstract

This project sets out to design and implement a mobile FTP client application for Android OS that accompanies a home media server using the Nas4Free operating system. The application utilizes Apache Commons' .net Java library to perform three functions: connect remotely to an FTP server, browse through directory listings, and download single files or entire directories to the Android device. This senior project encompasses multiple concepts including the configuration of a network to allow external access to a server behind a firewall, understanding of SSL/TLS security including private key encryption and self-signed certificates, the FTP protocol and its associated commands, and centers on Android development and the creation of an Android application. This application is for personal use only, and will not be released on the Google Play Store.

1 Introduction

1.1 Problem Statement

The idea for this project came from my use of high quality lossless audio files in FLAC format that are kept at home stored on a home media server. These files are quite large compared to standard MP3s, so I can only keep a limited number of songs on my phone at one time. Since the files I have stored at home are files I have created myself by ripping audio from CDs I own, a substitute does not exist that I could acquire legitimately, at least without paying for it. The server holding my files has a number of different services that allow sharing to devices over a local network or externally over the internet. This led me to understand that I could share my data with a mobile client by remotely connecting to my server and transferring the files I needed.

1.2 Solution

To solve this problem, an Android application must be created to act as the mobile client and transfer files to the device. The application should be created using the Android platform because it uses Java, which is widely used and easy to find libraries for, and because my phone runs Android which is the ultimate purpose of this app. Also, the app should utilize one of the remote transfer protocols included in the server's operating system. Since this app will be used on public networks, security is also a concern. The app should be able to establish an encrypted connection to the server to exchange login information privately, then navigate to the desired file or folder and transfer it to the device's local storage.

2 Frontend

2.1 Design

I decided to use FTPS as my networking solution because of its simplicity and security. The simplicity comes from the basic set of commands FTP uses to establish connection, navigate directories, and manage transfers. However, pure FTP lacks any security in its connection, transmitting sensitive login information in plain text and leaving data unencrypted, thus letting anyone between client and server to easily intrude. This problem is solved with the simple addition of TLS/SSL security into FTP's operation. TLS/SSL uses a combination of public/private key encryption along with a signed certificate to allow a secure connection between two devices.

After choosing what secure transfer protocol to use, it is time to select the proper library to simplify implementation of FTPS in Java. I chose the Apache Commons .net libraries because of their thorough documentation and continued support. Many other libraries have very little support because they are maintained by either small teams or individuals who often do the work for free, or do have support but require payment. Apache Commons satisfies both these requirements by being both free and supported by a large community of contributors.

Using the Apache Commons libraries revolves around a single FTPS object which maintains status, connection, settings, etc. The intricacies of Android programming prevent this common object from being passed between UI screens, so I

chose to use a global reference to make sure that the object would remain static and common through all the android code.

```
public class Globals extends Application {

    public static Object global_ftps;

}
```

Figure 1: Global object reference in Java

2.2 Establishing a Connection

Connecting remotely to the server requires a set of information that includes the server's host name or IP address, a username, and a password. Before sending this information, a secure connection must be established. This usually involves the exchange of a certificate which must be validated externally by a third party, or be accepted as valid by the user. However, due to the limitations of the chosen library this check is not performed, so caution should be taken when connecting to servers not owned by the user. Once the handshake is complete and traffic is encrypted, login information is then sent and a reply code shows if the connection was a success.

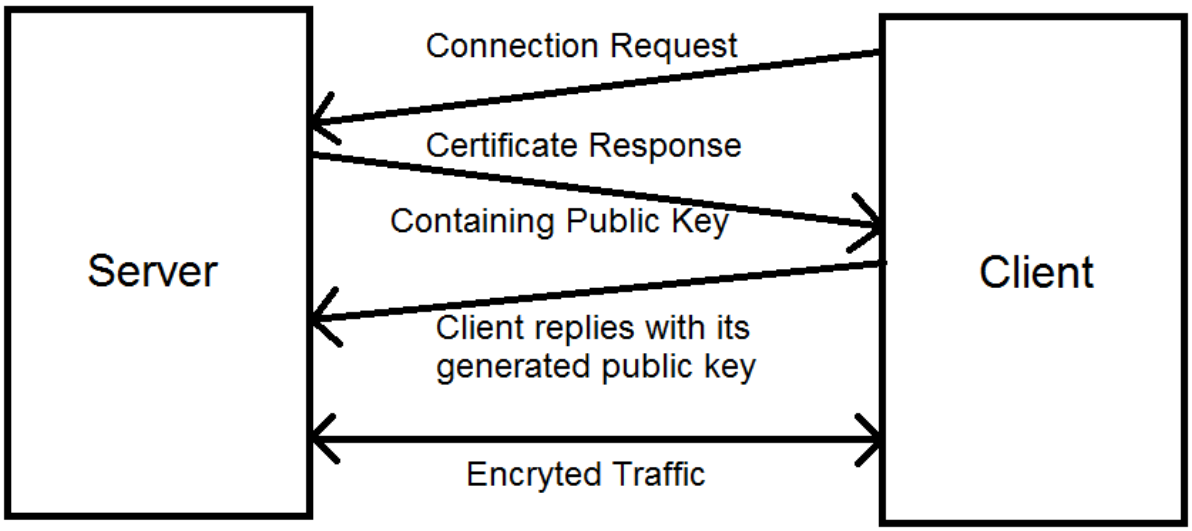

Figure 2: Process of setting up a secure connection

2.3 Permissions

After establishing a secure connection, certain settings must be modified so that a data link can be made. If these setting are configured improperly, the connection will be refused or terminated by the server. The first and most important setting is the PASV command which sets the FTP server to passive mode. This means that the server will reply with the port number to connect to for the data link. Next, to add protection to the data channel, the commands PROT for protection and PBSZ to set the protection buffer size to zero allow the data connection to be encrypted, which may be required by the server. Finally, compatibility with my server required setting the character encoding to UTF-8, but others systems may not.

```
ftps.setBufferSize(1000);

if (!ftps.login(username, password))
{
    ftps.logout();
    error = true;
    System.exit(1);
}

ftps.setFileType(FTP.BINARY_FILE_TYPE);
ftps.enterLocalPassiveMode();
ftps.sendCommand("OPTS UTF8 ON");
```

Figure 3: Login and settings commands

2.4 Directory listing

Now that the connection has been configured properly, the next step is navigating through the server's file system. This means listing the contents of given directories to see what files and folders lie within, so that the current working directory may be changed and files can be downloaded. In FTP, this is can be done with a few different commands. The NLST command returns just the names of the files and folders in a given directory in string format. The LIST command returns additional information on the specified file or directory, or the current directory if none is specified, in the form of an FTPFile object. The MLSD command returns the same information as the LIST command, but requires that you specify a directory. On some systems one or more of these commands may not be implemented, but for my implementation the LIST command sufficed.

```
fileNames = ftps.listNames();

files = ftps.listDirectories();
for (FTPFile file : files) {
    folders.add(file.getName());
}
```

Figure 4: Using the LIST and NLST commands

2.5 Transfer

After navigating to the file or folder the user wants to download, the transfer can be done one of two ways. The first option offers more control, where the download method returns an input stream that must be read from into a buffer and then written to an output file. This means that progress can be tracked, showing how much of the file has already been transferred which is very useful information for the user. The other option is easier, where the download method is passed an output stream and transfers the entire file all at once, dealing with any errors that come up. I chose the easier method so I did not have to worry about tracking progress, but further development necessitates adding progress tracking for the user's benefit.

```
out = new FileOutputStream(file);
if (ftps.retrieveFile(fname, out) == false) {
    System.err.println("Error downloading file");
    out.close();
    Intent i = new Intent(getApplicationContext(), Files.class);
    startActivity(i);
}
out.close();
```

Figure 5: Process of downloading a file

3 UI

3.1 Design

Android is at the core of the development of this application, providing the mobile platform to perform these remote transfers. Its widespread usage and accessibility means that an entire world of documentation, tutorials, and helpful advice exists online for people just getting started with android. Programming in Android is done entirely in Java and xml format and necessitates the use of an integrated development environment (IDE) to manage the structuring, compiling, running, and debugging of each project.

There are two major IDEs to be considered when starting Android development: Eclipse or Android Studio. I chose Eclipse because of what I had heard from my fellow developers and a recommendation from Dr. Smith. Using Eclipse requires the additional installation of the Android SDK for windows, which includes all the libraries required to program in Android as well as example projects, Android virtual device images, and debugging tools to assist in development. Once the workspace has been prepared, creating a new project results in a generic template that contains all the required resources needed to run a basic app. This includes required startup code, Android libraries, resources including image and layout files, and a project manifest containing important information about the app’s usage and structuring.

Android programming centers on the use of activities, which divide up the functionality of the application into different screens seen by the user. Each activity

serves its own purpose, whether it is gathering information through text boxes, showing items in a scrollable list, or just displaying loading screen while processing information. These activities consist of a body of code containing the default onCreate() method, which is called when starting or transitioning to the activity, and a layout file in xml detailing the look and function of the user interface. When an activity is launched, it is passed an Intent containing contextual information from the previous activity. This allows important data to be passed between screen transitions as well as entirely separate bodies of code. Project management can be found in the Android manifest, which contains a list of all activities and their usage filters, application information required by the Google play store such as version identification and the minimum/target Android version, and special permissions required for things like writing to memory and connecting to the internet.

The design of my app came from the process of downloading a file from a remote server, which involves connecting to the server and logging in, navigating directories to the desired file or folder, and then transferring it to the device. Thus, I created three activities to handle connection, file browsing, and downloading in independent screens. Dividing it up into these categories means simplifying the design by allowing each layout to be relatively static without the additional difficulty of including pop-up windows or complex animations.

3.2 Main Login Screen

The first activity opens by default when the app is first launched and handles acquiring of login information and connecting to the server. This title screen displays

three text boxes for entering the server address, username, and password, as well as a button to confirm the user has finished entering their information. These text boxes are editable text views, which allow for default text to show the user what information they should enter and where. Once the user has finished and pressed the button, a callback function associated with the button click action begins executing the FTPS code to connect and login to the server. If the connection is accepted the FTPS object is saved to a global state and the File Listing activity is started, otherwise the app stays on the same screen and prompts the user to re-enter their information.

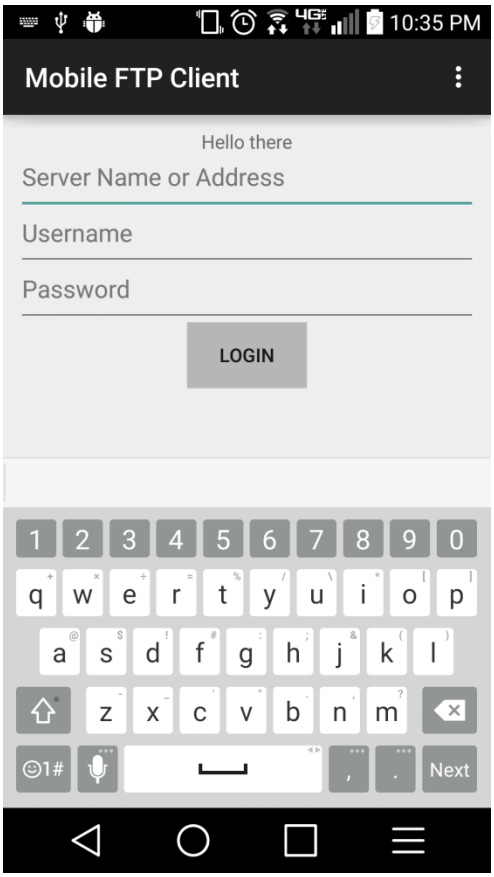

Figure 6: Login screen

3.3 File Listing Screen

When navigating through a series of directories, two things are necessary to know. First is the path for the current working directory and second, the contents of that directory. I put the working directory's path at the top of the screen as a header to the list so that it is easily visible and always present. This is done using a standard text view whose text is set dynamically each time the activity is launched. The contents of the directory are presented in a scrollable list view, where each item can be either pressed regularly or long pressed, depending on if you wish to change the directory or download that file/folder respectively. The list view is dynamically filled by an array of strings provided by the LIST method using an array adaptor which articulates each datum in the array with its own clickable list item. When the user clicks on a folder, the change directory command is sent and the Files activity is relaunched so that the screen can be repopulated with the contents of the new directory. If the user wants to go back to a previous directory, pressing the back button changes the working directory into the parent directory and then relaunches the Files activity. This is accomplished by overriding the method that intercepts the back button press, which normally returns to the activity the current activity was launched from. Once the user has found the file or folder they want to download, a long press causes the name of the file or folder and a flag to specify whether it is a directory to be stored in an intent before launching the Download activity.

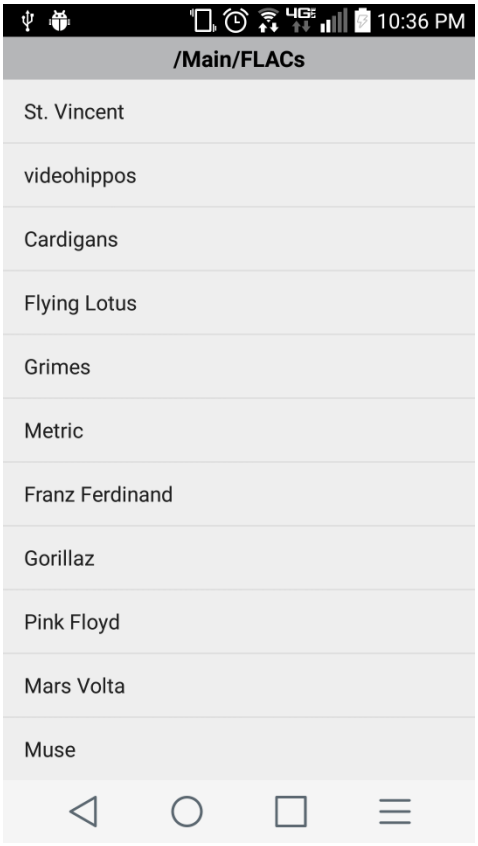

Figure 7: File listing

3.4 Download Screen

After launching the Download activity, the chosen file or folder name is displayed in a question asking if the user is sure they want to commence the download. This prevents any accidental transfers from wrong or unintentional clicks. The two buttons labeled yes and no allow the user to either continue and start the transfer or return to browsing through files, which relaunches the Files activity. Pressing the yes button starts the transfers and removes the confirmation text and buttons, replacing them with text confirmations as each item finishes downloading showing the user's progress if transferring multiple files. This is accomplished by accessing the layout, button, and text view resources through their given ids and modifying the layout by first removing the

current buttons and text views, and then replacing them with dynamically created text views to allow varying amounts depending on the number of files being transferred. Once all the transfers have finished, a button appears prompting the user to continue which launches the Files activity and returns them to the current working directory.

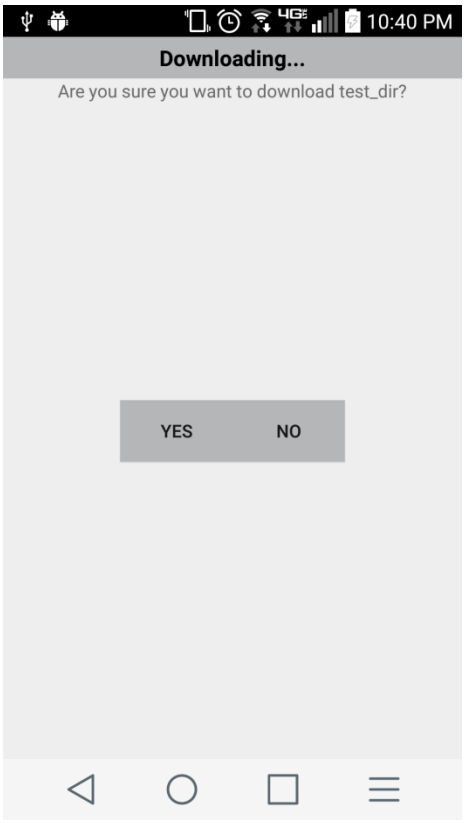

Figure 8: Confirmation prompt

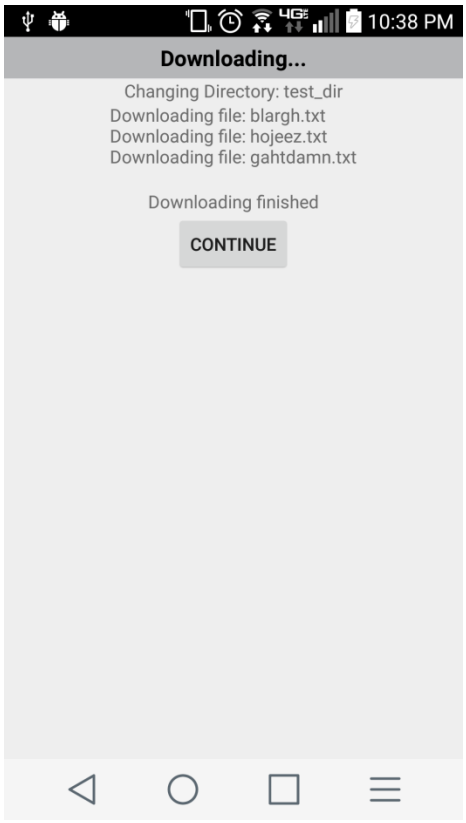

Figure 9: Transfer completed

4 Server

4.1 Configuration of Server

The server I used for this project is a Linux and FreeBSD based freeware operating system call Nas4Free designed to make it easy for a layperson to install and setup a server using an old PC. The server uses a web console to configure any number of different networking and sharing services such as SSH, FTP, drive sharing for both Apple and Windows computers, and even web hosting. When setting up FTP a plethora of options are available to configure almost every aspect of its operation, but security is the main concern since it will be accessed from outside of my home network.

Two things are required to enable SSL/TLS connections, a signed certificate and a private key. The process for generating these yourself is pretty straight forward, using a command prompt and the free utility openssl. The first step is to create the private key, where you specify the bit length of the key to be generated and openssl stores it in PEM format, which means it is readable as ASCII text. After obtaining the private key, openssl uses that key in conjunction with information queried about owner or operator of the server to generate a certificate signing request (CSR). The new CSR needs to be signed by an authority, otherwise it will generate an error to the user and require them to accept it as valid if they wish to continue and connect to the server, but this is not an issue if it is not used externally. To sign the CSR, opennssl requires the duration of validity for the certificate and a key to sign it with, which can be the same as the key created earlier. The newly generated certificate is in x509 PEM format, which is also

readable as ASCII text, and both the text for the certificate and the private key should be copied into their respective text boxes when enabling SSL/TLS for FTPS.

4.2 Configuration of Router

After enabling secure connections, the router needs to be configured to allow incoming connections from outside the network. This can be accomplished a couple of different ways involving a varying level of risk to the security of the local network. Port forwarding, which sends all traffic destined for a specific port to an associated address on the local network, can be used to limit the traffic allowed through the firewall to ports that are adequately protected on the receiving end. Similarly, port triggering allows forwarded ports to be created based on ports used for outbound traffic. Unfortunately, I could not configure my router to get either of these methods to work properly. This left one remaining option, setting up a DMZ for my server. Setting up a DMZ means that all inbound traffic is forwarded to one IP address on the network, and this can be extremely dangerous. Allowing all the ports on a device to be accessible from the Internet means that anyone can probe the device for holes in its security, whether that means exploiting known vulnerabilities in an old version of software or using brute force to crack a password. This may have worked for me as a temporary option, but opening the network in this way leaves it susceptible to an inevitable attack.

5 Testing

5.1 FileZilla

To begin testing from the bottom up, I had to start with something I knew already worked. This meant using a widely used and fully supported FTP/FTPS client such as FileZilla. When configuring the server, FileZilla allowed me to test whether my newly generated private key and certificate set function properly, which it did not initially. I discovered that FTPS service was not running because my certificate was encrypted with a password, which was easily removed. Most importantly, FileZilla helped me debug a complex issue where I could connect to the server and login but not open data connections to retrieve directory listings. I used Wireshark, a packet sniffing utility, in conjunction with FileZilla to find the exact sequence of commands sent to the server, since FileZilla worked and my java implementation of FTPS did not. This is how I found out the optional UTF-8 setting was required for my server.

5.2 Pure Java FTPS

Before combining FTPS library code with Android, I created an example program to prove that Apache Commons' FTPS code worked with my server and to acquaint myself with the process and methods needed to emulate FileZilla's operation. Using this test program, I was able to try out each of the methods both independently and in conjunction with others. This let me find the correct sequence of commands to connect to the server and configure its settings. Also, testing multiple methods that performed a

similar function, such as the LIST, NLST, and MLSD commands, meant I could choose one that best fulfilled the required functionality.

5.3 Android with FTPS

Testing for Android began with a tutorial that went over basic app development, so I could figure out how each of the components that would later make up the app worked before having to worry about FTPS. Through this tutorial, I was able to test and learn incrementally, adding features and discussing concepts one at a time. After becoming familiar with each of the different modules like buttons, text views, and layouts, it was easy to recombine them into what I needed without running into any errors. Also, since the FTPS code had also already been tested independently, it was simple to copy the correct series of methods and articulate them with the Android environment. Using Eclipse with the Android SDK allowed me the option to test the app on both an Android virtual device as well as my own personal phone. Testing with both a virtual phone using just my computer, removing the hassle of setup or risk of using my own phone, and with actual hardware that would be the target device meant faster development and more robust debugging. Running the app on my own phone brought to light certain issues I couldn't have done with just a virtual device, such as where certain files are actually stored. Eclipse and the Android SDK also provide a couple additional ways to help with testing. Android devices connected to Eclipse have a live stream of logged message displayed, which can be printed to using Android's built in message logger or just using standard out, and is useful for printf debugging and seeing what the Android OS is doing in the background. Java's own ability to print a stack trace when encountering errors helped me immensely as well.

6 Related Work

6.1 AndFTP

AndFTP is a professionally developed Android mobile app that works as an FTP client and supports SSL/TLS connections as well as additional features for extended support and syncing of directories to or from the server. I intended to use this app as a functional reference to see how FTPS worked on mobile. Unfortunately, I never got it to work with my server, so it instead provided further reason for my development of this app.

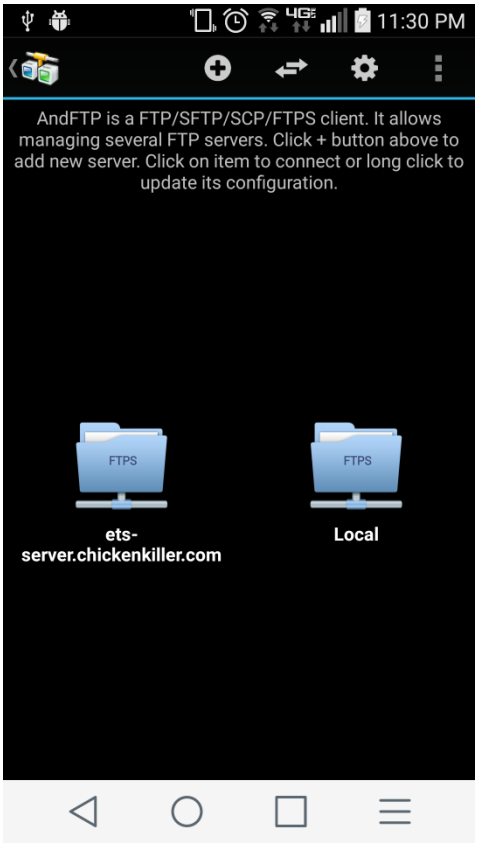

Figure 10: AndFTP screen capture

6.2 FileZilla

FileZilla is a popular free cross-platform FTP client and/or server that features a well-designed and easy to use interface, full support of almost every facet of FTP, useful integration with operating system utilities, and much more. I used FileZilla to test if the server was properly configured after setting up SSL/TLS, making sure I had generated my private key and certificate correctly. Also, it let me see the flow of FTP commands required to properly create new connections.

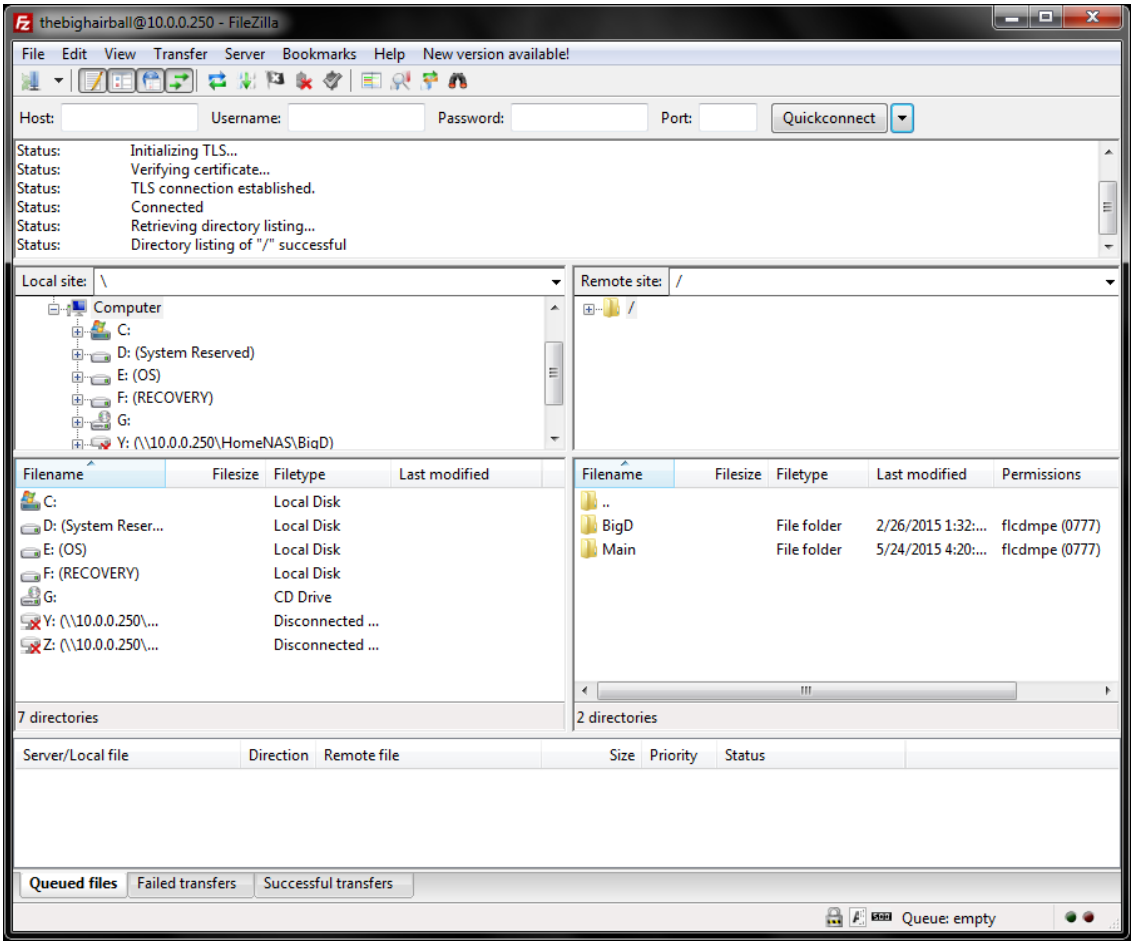

Figure 11: FileZilla's user interface

6.3 Nas4Free

Nas4Free is an open source network-attached storage (NAS) operating system based on FreeBSD that supports sharing across multiple platforms using a multitude protocols. It was previously being used for sharing drives on my local network using Samba and AFP, but was easily configured to function additionally as a FTP server. All options are accessible through a web interface that also provides information about hardware usage and utilities for formatting and monitoring volumes attached to the system.

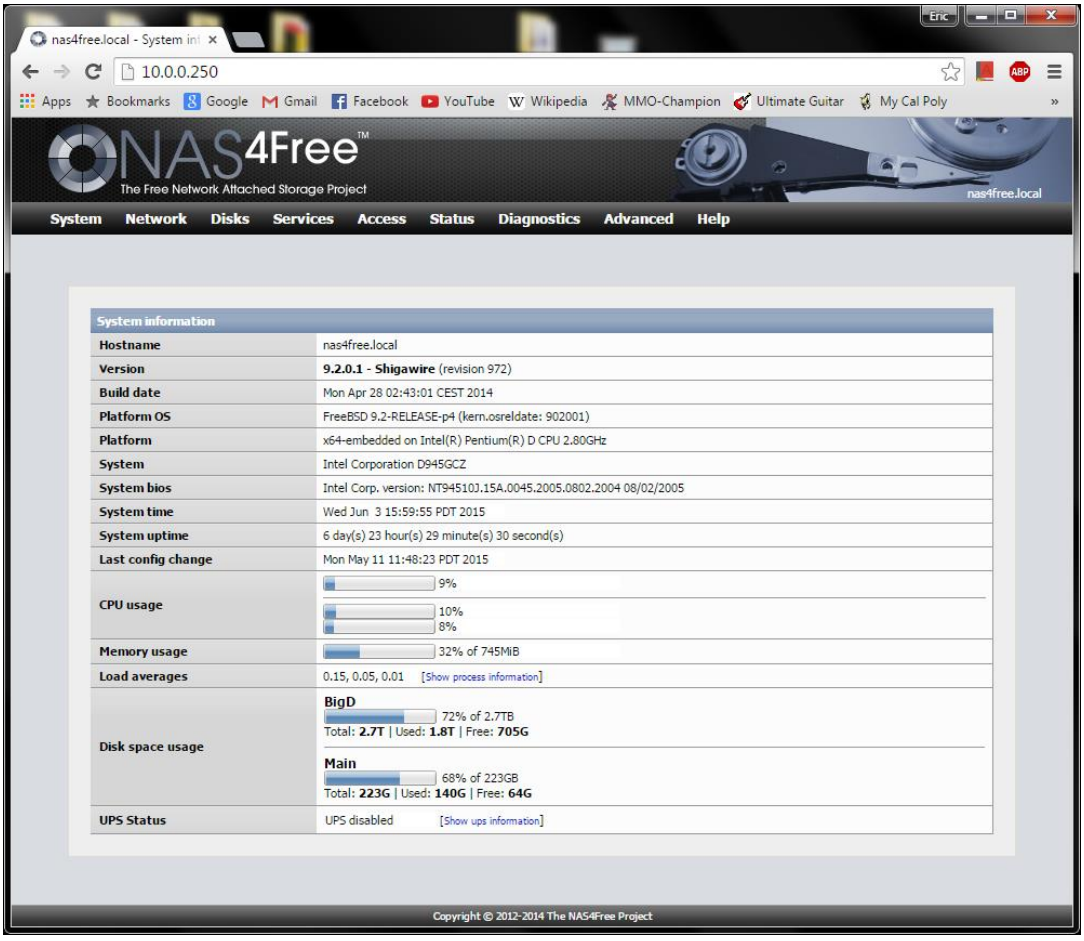

Figure 12: Nas4Free's web interface

7 Conclusion

This project has provided me an opportunity to learn about Android and app development, and I feel I have gained a better understanding of the Java programming language as well. Having to learn Android programming completely from scratch was difficult at times, but I feel I have become a better and more well-rounded programmer for my efforts. Working on this project during and after taking Introduction to Computer Networks has really aided my understanding of network security and the dangers of opening your home network to possible intrusions. For future reference, I would advise against allowing any holes in a home network's security unless you can be absolutely certain that whatever receives that external traffic is sufficiently protected.

The app in its current state is more of a functional prototype than a finished product. It does all of what it was originally intended to do, but not in the best or most efficient matter. Currently, all of the networking functions run in the main thread which can prevent the user interface from updating or responding to user input. In addition to moving these functions into an asynchronous process, I would have implemented background transfers so that downloading could continue if the user switches to another app. Much more time could be spent on making the user interface look more appealing, but the app fulfills its purpose and I feel satisfied in what I have accomplished in the time I have been given.

8 References

1. *Eclipse*

   Available at: https://eclipse.org/

2. *Android SDK*

   Available at: http://developer.android.com/sdk/index.html

3. *FileZilla*

   Available at: https://filezilla-project.org/download.php

4. *AndFTP*

   Available at: https://play.google.com/store/apps/details?id=lysesoft.andftp&hl=en

5. *Nas4Free*

   Available at: http://www.nas4free.org/

9 Analysis of Senior Project Design

Summary of Functional Requirements

The functionality contained in this project consists of connecting to a remote FTP server through an Android mobile device, browsing through directory listings on the server, and downloading single files or entire directories to the Android device.

Primary Constraints

Significant challenges encountered over the course of this project came in two forms: the first was having to learn Android programming from scratch with only some latent experience in Java, and the second was configuring my network to allow external connections to the server.

Economic

Since this is a software project, the economic costs are minimal. Equipment costs include approximately $1100 for the laptop for development, $600 for the testing phone, and $400 for the repurposed PC used as a media server, totaling around $2100. Estimated development time was about 100 hours, but actual development time came out to be closer to 70-80 hours.

Environmental

There is no inherent environmental impact associated with this project aside from external costs such as construction and power consumption of devices used to develop or run the application.

Manufacturability

There is no manufacturing involved with software and distribution is as easy as downloading a file over the internet.

Sustainability

The only concern regarding sustainability would be power consumption during development and usage. As far as continued development of the product's lifespan, plenty of additional features can still be added as mentioned earlier in the report, and the only challenge associated with upgrading this software is my own sloppy development hindering later revisions.

Ethical

Whenever you allow people a platform to share files amongst each other, you run the risk of enabling the distribution of copyrighted works. Unfortunately, it is extremely difficult to regulate this sort of behavior, so responsibility lies with the user to help prevent piracy.

Health and Safety

An issue concerning safety for this project is regarding security when using a server connected to a home network. Allowing remote connections to enter through your router's firewall and access the server can mean creating dangerous holes in your networks security. This means potential intruders can probe these holes for vulnerabilities and possibly compromise the entire network through just the server.

Social and Political

There are no social or political concerns regarding this project, with the exception of using publicly licensed code that must be properly attributed and not distributed for profit.

Development

During the development of this project, I used a multitude of new tools and technologies and learned a great deal about them. To develop in Android, I used Eclipse for the first time and in conjunction with the ADK plugin required to program and test Android apps. I also used FileZilla to interface with my media server using FTP. Through both a tutorial and independent research I learned the structure and process associated with designing and building an Android application.

10 Appendix

MainActivity.java

```
package com.example.myfirstapp;

import java.io.IOException;
import java.io.PrintWriter;

import org.apache.commons.net.PrintCommandListener;
import org.apache.commons.net.ftp.FTP;
import org.apache.commons.net.ftp.FTPConnectionClosedException;
import org.apache.commons.net.ftp.FTPReply;
import org.apache.commons.net.ftp.FTPSClient;

import android.content.Intent;
import android.os.Bundle;
import android.os.Environment;
import android.os.StrictMode;
import android.support.v7.app.ActionBarActivity;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;


public class MainActivity extends ActionBarActivity implements OnClickListener {

   private EditText server, username, password;
   private Button theButton;

   boolean error = false;
   String protocol = "SSL";
   FTPSClient ftps;

   public static final String MY_APP_PREFS = "MyAppPrefs";

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      StrictMode.ThreadPolicy policy = new
            StrictMode.ThreadPolicy.Builder().permitAll().build();
      StrictMode.setThreadPolicy(policy);

      ftps = new FTPSClient(protocol);
      ftps.addProtocolCommandListener(new PrintCommandListener(new
            PrintWriter(System.out)));
```

```
   server = (EditText) findViewById(R.id.server);
   username = (EditText) findViewById(R.id.username);
   password = (EditText) findViewById(R.id.password);

   theButton = (Button)findViewById(R.id.myButton);
   theButton.setOnClickListener(this);
}

public void onClick(View v) {
   //respond to click
   if (v.getId() == theButton.getId()) {

      // init connection
      try
      {
         int reply;

         ftps.connect(server.getText().toString());
         System.out.println("Connected to " + server.getText().toString() + ".");

         reply = ftps.getReplyCode();

         if (!FTPReply.isPositiveCompletion(reply))
         {
            ftps.disconnect();
            System.err.println("FTP server refused connection.");
            System.exit(1);
         }
      }
      catch (IOException e)
      {
         if (ftps.isConnected())
         {
            try
            {
               ftps.disconnect();
            }
            catch (IOException f)
            {
               // do nothing
            }
         }
         System.err.println("Could not connect to server.");
         e.printStackTrace();
         System.exit(1);
      }

      // log in
      try
      {
         ftps.setBufferSize(1000);

         if (!ftps.login(username.getText().toString(),
               password.getText().toString()))
         {
```

```
                ftps.logout();
                error = true;
                System.exit(1);
            }

            ftps.setFileType(FTP.BINARY_FILE_TYPE);
            ftps.enterLocalPassiveMode();
            ftps.sendCommand("OPTS UTF8 ON");

            Globals.global_ftps = ftps;

            Intent i = new Intent(getApplicationContext(), Files.class);
            startActivity(i);
        }
        catch (FTPConnectionClosedException e)
        {
            error = true;
            System.err.println("Server closed connection.");
            e.printStackTrace();
        }
        catch (IOException e)
        {
            error = true;
            e.printStackTrace();
        }

        System.out.println(Environment.getDataDirectory());
        System.out.println(Environment.getExternalStorageDirectory());
    }
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();
    if (id == R.id.action_settings) {
        return true;
    }
    return super.onOptionsItemSelected(item);
}
}
```

Files.java

```
package com.example.myfirstapp;

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

import org.apache.commons.net.ftp.FTPFile;
import org.apache.commons.net.ftp.FTPSClient;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;

public class Files extends Activity {

   FTPSClient ftps;
   ArrayList<String> folders = new ArrayList<String>();

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_files);

      String[] fileNames = null;
      FTPFile[] files;

      ftps = (FTPSClient)Globals.global_ftps;

      // get pwd and file listing
      try
      {
         String pwd = ftps.printWorkingDirectory();
         if (pwd != null) {
            TextView myTextView = (TextView) findViewById(R.id.workindir);
            myTextView.setText(pwd);
         }

         fileNames = ftps.listNames();

         files = ftps.listDirectories();
         for (FTPFile file : files) {
            folders.add(file.getName());
         }
      }
      catch (IOException e)
```

```
{
   e.printStackTrace();
   Intent i = new Intent(getApplicationContext(), MainActivity.class);
   startActivity(i);
}

final ListView listview = (ListView) findViewById(R.id.listview);

final ArrayList<String> list = new ArrayList<String>();
for (int i = 0; i < fileNames.length; ++i) {
   list.add(fileNames[i]);
}
final StableArrayAdapter adapter = new StableArrayAdapter(this,
                    android.R.layout.simple_list_item_1, list);
listview.setAdapter(adapter);

listview.setOnItemClickListener(new AdapterView.OnItemClickListener() {

   @Override
   public void onItemClick(AdapterView<?> parent, final View view,
                           int position, long id) {
      final String item = (String) parent.getItemAtPosition(position);
      view.animate().setDuration(1000).alpha((float)0.5)
                    .withEndAction(new Runnable() {
         @Override
         public void run() {

            for (String folder : folders) {
               if (item.equals(folder)) {
                  try
                  {
                     ftps.cwd(item);
                  }
                  catch (IOException e)
                  {
                     e.printStackTrace();
                     Intent i = new Intent(getApplicationContext(),
                                           MainActivity.class);
                     startActivity(i);
                  }

                  Intent i = new Intent(getApplicationContext(), Files.class);
                  startActivity(i);
               }
            }
         }
      });
   }

});

listview.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {

   @Override
   public boolean onItemLongClick(AdapterView<?> parent, final View view,
```

```
                                int position, long id) {
            final String item = (String) parent.getItemAtPosition(position);
                           view.animate().setDuration(1000).alpha((float)0.5)
                           .withEndAction(new Runnable() {
               @Override
               public void run() {

                  Intent i = new Intent(getApplicationContext(), Download.class);

                  i.putExtra("file", item);

                  for (String folder : folders) {
                     if (item.equals(folder)) {
                        i.putExtra("isDir", true);
                     }
                  }

                  startActivity(i);
               }
            });

            return true;
         }

      });
   }

   @Override
   public void onBackPressed() {
      try
      {
         ftps.changeToParentDirectory();
      }
      catch (IOException e)
      {
         e.printStackTrace();
         Intent i = new Intent(getApplicationContext(), MainActivity.class);
         startActivity(i);
      }

      Intent i = new Intent(getApplicationContext(), Files.class);
      startActivity(i);
   }

   private class StableArrayAdapter extends ArrayAdapter<String> {

      HashMap<String, Integer> mIdMap = new HashMap<String, Integer>();

      public StableArrayAdapter(Context context, int textViewResourceId,
                           List<String> objects) {
         super(context, textViewResourceId, objects);
         for (int i = 0; i < objects.size(); ++i) {
            mIdMap.put(objects.get(i), i);
         }
      }
```

```
      @Override
      public long getItemId(int position) {
         String item = getItem(position);
         return mIdMap.get(item);
      }

      @Override
      public boolean hasStableIds() {
         return true;
      }
   }
}
```

Download.java

```
package com.example.myfirstapp;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;


import java.util.ArrayList;

import org.apache.commons.net.ftp.FTPFile;
import org.apache.commons.net.ftp.FTPSClient;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.os.Environment;
import android.view.View;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.LinearLayout.LayoutParams;
import android.widget.TextView;

public class Download extends Activity {

   FTPSClient ftps;
   Boolean isDir = false;
   String fname = null;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_download);

      ftps = (FTPSClient)Globals.global_ftps;
      Intent myIntent = getIntent();
      fname = myIntent.getStringExtra("file");
```

```
    isDir = myIntent.getBooleanExtra("isDir", false);

    TextView tv = (TextView) findViewById(R.id.yousure);
    tv.setText("Are you sure you want to download " + fname + "?");

    Button btn_yes = (Button) findViewById(R.id.button_yes);
    btn_yes.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View v) {
            LinearLayout parent = (LinearLayout) findViewById(R.id.download_layout);
            View child = (View) findViewById(R.id.button_bar);
            parent.removeView(child);

            downloadFiles();
        }
    });

    Button btn_no = (Button) findViewById(R.id.button_no);
    btn_no.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View v) {
            Intent i = new Intent(getApplicationContext(), Files.class);
            startActivity(i);
        }
    });
}

private void downloadFiles() {

    String[] fileNames = null;
    ArrayList<String> folders = new ArrayList<String>();
    FTPFile[] files;

    // Layout/view stuff
    LinearLayout linlay = (LinearLayout) findViewById(R.id.download_layout);
    LayoutParams lparams = new LayoutParams(
          LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);
    TextView tv = (TextView) findViewById(R.id.yousure);
    tv.setText("tmp");

    File path = Environment.getExternalStoragePublicDirectory(
                                    Environment.DIRECTORY_DOWNLOADS);

    if (!isDir) {
        File file = new File(path, fname);
        FileOutputStream out;

        tv.setText("Downloading file: " + fname);

        try {
            out = new FileOutputStream(file);
            if (ftps.retrieveFile(fname, out) == false) {
                System.err.println("Error downloading file");
```

```
            out.close();
            Intent i = new Intent(getApplicationContext(), Files.class);
            startActivity(i);
         }
         out.close();
      }
      catch (IOException e) {
         e.printStackTrace();
         Intent i = new Intent(getApplicationContext(), Files.class);
         startActivity(i);
      }
   }
   else {
      File directory = new File(path, fname);
      FileOutputStream out;

      tv.setText("Changing Directory: " + fname);

      if (!directory.exists()) {
         directory.mkdir();
      }

      //get files and directories
      try
      {
         ftps.cwd(fname);

         fileNames = ftps.listNames();

         files = ftps.listDirectories();
         for (FTPFile file : files) {
            folders.add(file.getName());
         }
      }
      catch (IOException e)
      {
         e.printStackTrace();
         Intent i = new Intent(getApplicationContext(), Files.class);
         startActivity(i);
      }

      TextView textView2 = new TextView(this);
      textView2.setLayoutParams(lparams);
      textView2.setText("");
      linlay.addView(textView2);

      isDir = false;
      for (String f : fileNames) {
         for (String folder : folders) {
            if (f.equals(folder)) {
               isDir = true;
            }
         }
         if (!isDir) {
            textView2.append("Downloading file: " + f + "\n");
```

```
                try {
                   out = new FileOutputStream(new File(directory, f));
                   if (ftps.retrieveFile(f, out) == false) {
                      System.err.println("Error downloading file");
                      out.close();
                      Intent i = new Intent(getApplicationContext(), Files.class);
                      startActivity(i);
                   }
                   out.close();
                }
                catch (IOException e) {
                   e.printStackTrace();
                   Intent i = new Intent(getApplicationContext(), Files.class);
                   startActivity(i);
                }
             }
             isDir = false;
          }

          try
          {
             ftps.changeToParentDirectory();
          }
          catch (IOException e)
          {
             e.printStackTrace();
             Intent i = new Intent(getApplicationContext(), MainActivity.class);
             startActivity(i);
          }
       }

       TextView textView3 = new TextView(this);
       textView3.setLayoutParams(lparams);
       textView3.setText("Downloading finished");
       linlay.addView(textView3);

       Button btn = new Button(this);
       btn.setId(666);
       btn.setText("Continue");
       linlay.addView(btn, lparams);
       btn.setOnClickListener(new View.OnClickListener() {
          public void onClick(View v) {
             Intent i = new Intent(getApplicationContext(), Files.class);
             startActivity(i);
          }
       });
    }
}
```

activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center_horizontal"
    android:orientation="vertical"
    android:layout_margin="10dp" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello" />

    <EditText
       android:id="@+id/server"
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       android:hint="@string/server_info"
       android:inputType="text" />

    <EditText
       android:id="@+id/username"
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       android:hint="@string/username_info"
       android:inputType="text" />

    <EditText
       android:id="@+id/password"
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       android:hint="@string/password_info"
       android:inputType="text" />

    <Button
        android:id="@+id/myButton"
        android:onClick="ButtonClicked"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/click"
        android:background="#b5b6b8"
        android:padding="5dp" />

</LinearLayout>
```

activity_files.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/workindir"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:clickable="false"
        android:longClickable="false"
        android:paddingBottom="5dip"
        android:paddingTop="5dip"
        android:background="#b5b6b8"
        android:text="@string/nodir"
        android:textColor="#000000"
        android:textSize="17sp"
        android:textStyle="bold" />

    <ListView
        android:id="@+id/listview"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />


</LinearLayout>
```

activity_download.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/download_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/downloading"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:clickable="false"
        android:longClickable="false"
        android:paddingBottom="5dip"
```

```
        android:paddingTop="5dip"
        android:background="#b5b6b8"
        android:text="@string/downloading"
        android:textColor="#000000"
        android:textSize="17sp"
        android:textStyle="bold" />

    <TextView
        android:id="@+id/yousure"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/areyousure" />

    <LinearLayout
        android:id="@+id/button_bar"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:orientation="horizontal" >

        <Button
            android:id="@+id/button_yes"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/yes"
            android:background="#b5b6b8"
            android:padding="5dp" />

        <Button
            android:id="@+id/button_no"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/no"
            android:background="#b5b6b8"
            android:padding="5dp" />
    </LinearLayout>

</LinearLayout>
```

Android Manifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myfirstapp"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="16"
        android:targetSdkVersion="22" />
```

```
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".Files"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />

                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
        <activity
            android:name=".Download"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />

                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```