**SPL-1 Project Report, 2019**

**MUSIC HUNTER**

**Course: Software Project Lab I**

**Course No: SE 305**

**Submitted by**

***MD. Billal Hosain Nirab***

**BSSE Roll No. : 1020**

**BSSE Session: 2017-2018**

**Supervised by**

***Nadia Nahar***

**Designation: Lecturer**

**Institute of Information Technology**



**Institute of Information Technology**

**University of Dhaka**

**29-05-2019**

**Table of Contents**

**1. Introduction 04**

**1.1 Background Study 04**

**1.1.1 Morphological Image Processing 05**

**1.1.1.1 Implementation of Erosion 06**

**1.1.1.2 Implementation of Dilation 08**

**1.1.2 Message Encryption and Decryption (WAV file Specification) 10**

**1.1.3 SHA-1 12**

**1.2 Challenge 16**

**2. Project Overview 19**

**2.0 Scan the QR code to watch the full features of MUSIC HUNTER 20**

**2.1 Import multiple Song at a time and make a playlist 20**

**2.2 Play song from playlist 20**

**2.2.1 Manually 20**

**2.2.2 Using Gesture 22**

**2.3 ADD song into favorite list 22**

**2.3.1 Manually 22**

**2.3.2 Automatically detect weather a song is liked by the user or not and**

**add it to favorite list 23**

**2.3.3 Automatically remove song from favorite list if user doesn’t listen it**

**certain period of time 25**

**2.4 User Account 26**

**2.4.1 Create Account (password is encrypted using SHA-1) 26**

**2.4.2 Login 26**

**2.5 Server 26**

**2.6 Message Encryption and Decryption 28**

**2.6.1 Encryption 28**

**2.6.2 Decryption 29**

**3. User Manual 30**

**3.1 Create Account 30**

**3.2 Login 31**

**3.3 Import song into playlist 31**

**3.4 Play song from playlist 33**

**3.4.1 Manually 33**

**3.4.2 using Gesture 34**

**3.5 Add song into favorite list 35**

**3.5.1 Manually 35**

**3.5.2 Automatically 35**

**3.6 Message Encryption and Decryption 35**

**3.6.1 Encryption 35**

**3.6.2 Decryption 36**

**4. Conclusion 36**

**5. Appendix 37**

**References 37**

1. **Introduction**

Introducing a Music Player which is not like a traditional music player application. Irrelevant talk aside, the reason why I'm calling it that it is far away from traditional music player application and more specialized because of its following features.

It has option to import music into playlist and user can play those songs from playlist. But this is not enough to call it special. It has option to control music using gesture. If a user swipe up and down he/she can control volume. By swiping up the volume is increasing and by swiping down the volume is decreasing. And by swiping left and right user can change song. By swiping left he/she will be able to play previous song and by swiping right he/she will be able to play the next song.

Sometimes we listen a song thousand times but still we forget to add it into our favorite list. So that if we want to listen it further then we faced some trouble to find that song. But don’t worry about it. Music Hunter can solve this problem. It has ability to track our favorite song and automatically add it into favorite list. User can also put a song into favorite list by set a rating of a song.

There is no chance for losing user playlist because of user can create an account along with a private playlist and the whole playlist and user info are stored into the MUSIC HUNTER server.

It is mentioned that the user account is fully secured in the server because password of a user are stored in the server using SHA-1 cryptographic algorithm.

It has a server to handle user account and user can encrypt and decrypt their message by pushing message into audio data and no one can recognize the change of an audio.

Finally we call it a **MUSIC HUNTER.**

* 1. **Background study**
     1. **Morphological Image Processing**

**1.1.1.1 Erosion**

**1.1.1.2 Dilation**

* + 1. **Message Encryption and Decryption**

**WAV file specifications**

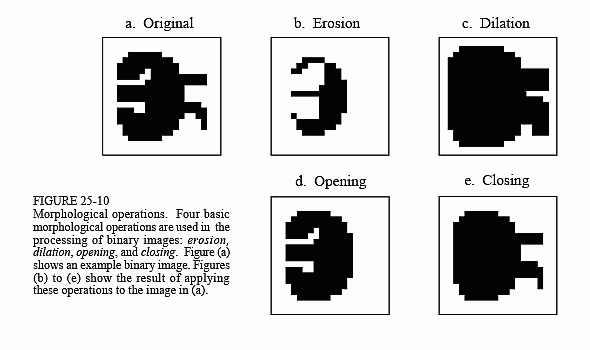
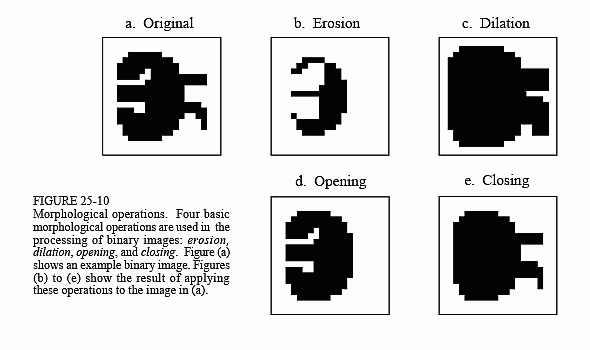
* + 1. **Cryptographic Hash algorithm**

**SHA-1**

**1.1.1 Morphological Image Processing**

For gesture recognition I have to capture frame using webcam. For each frame I am seeking to find the RED color. Using RED color I am trying to track the movement of an object in front of the camera. So that, for better tracking result I have to do morphological operation. So what is morphological operation?

The identification of objects within an image can be a very difficult task. One way to simplify the problem is to change the grayscale image into a binary image, in which each pixel is restricted to a value of either 0 or 1. The techniques used on these binary images go by such names as: blob analysis, connectivity analysis, and morphological image processing (from the Greek word morphē, meaning shape or form). The foundation of morphological processing is in the mathematically rigorous field of set theory; however, this level of sophistication is seldom needed. Most morphological algorithms are simple logic operations and very ad hoc.



In other words, each application requires a custom solution developed by trial-and-error. This is usually more of an art than a science. A bag of tricks is used rather than standard algorithms and formal mathematical properties. Here are some examples.

For example from figure (a) Each pixel in the background is displayed as white, while each pixel in the object is displayed as black. Frequently, binary images are formed by thresholding a grayscale image. Pixels with a value greater than a threshold are set to 1, while pixels with a value below the threshold are set to 0. It is common for the grayscale image to be processed with linear techniques before the thresholding.

**Erosion and Dilation**

Figures (b) and (c) show how the image is changed by the two most common morphological operations, erosion and dilation. In erosion, every object pixel that is touching a background pixel is changed into a background pixel. In dilation, every background pixel that is touching an object pixel is changed into an object pixel. Erosion makes the objects smaller, and can break a single object into multiple objects. Dilation makes the objects larger, and can merge multiple objects into one.

**1.1.1.1 Implementation of Erosion**

To implement erosion first we need to make a 3\*3 matrix which represent structuring element. For erosion, make all of the pixel of structuring element white.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

Then travers the structuring element on the target image. If any of the pixel of targeted image matches with any of the pixel of structuring element then copy top left corner of the structuring element pixel into targeted image. And do it repeatedly until structuring element travers the full image. For example :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

Figure (a) Figure (b) figure (c)

Here figure (b) is a 3\*3 structuring element and all pixel of structuring element is white. In figure (a) we have a binary image. From original RGB image we produce a binary image by applying threshold. To apply threshold first apply grayscale effect on it and then travers full image and if a pixel in white or approximately white then make it completely white and if a pixel is black or approximately black then make it completely black.

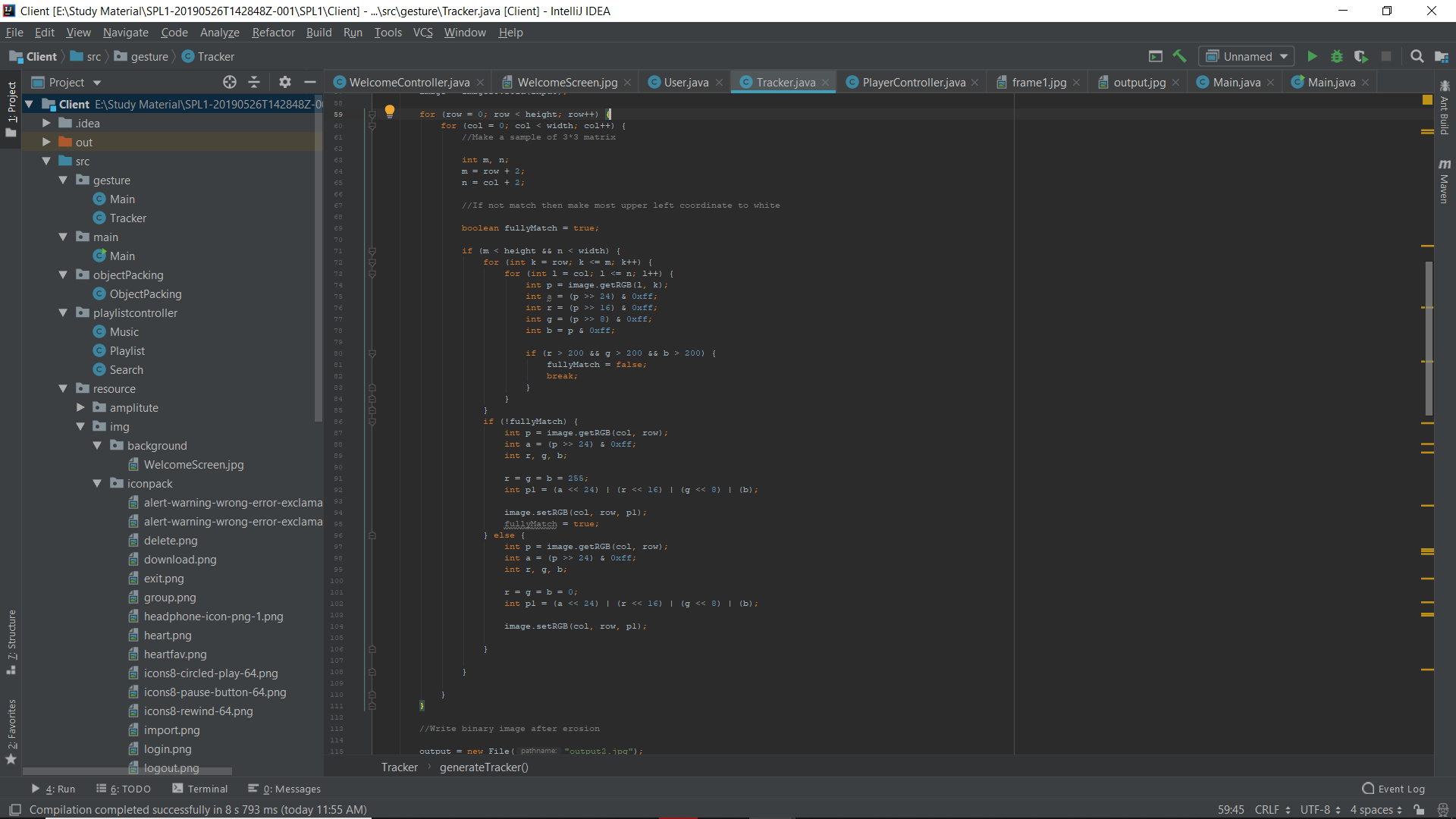


Figure (d)

After applying threshold effect then apply erosion to targeted image. Here figure (d) showing that how to implement erosion on an image.

**1.1.1.2 Implementation of Dilation**

Dilation is opposite operation of erosion. To implement dilation first we have to make 3\*3 matrix which represent structuring element same as erosion and make all pixel of structuring element to black. After applying threshold from grayscale image which grayscale image comes from RGB image then we have to apply dilation operation.

In dilation if any of the pixel of targeted image matches with structuring element then make top right corner of the targeted image black and do it repeatedly until structuring element travers full image.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

Figure (a) Figure (b) Figure (c)

Here figure (a) show the threshold image which produce from original image which is captured by webcam and figure (b) is 3\*3 structuring element and after dilation we produce an image which is shown in figure (c).

The implementation of dilation is given bellow.

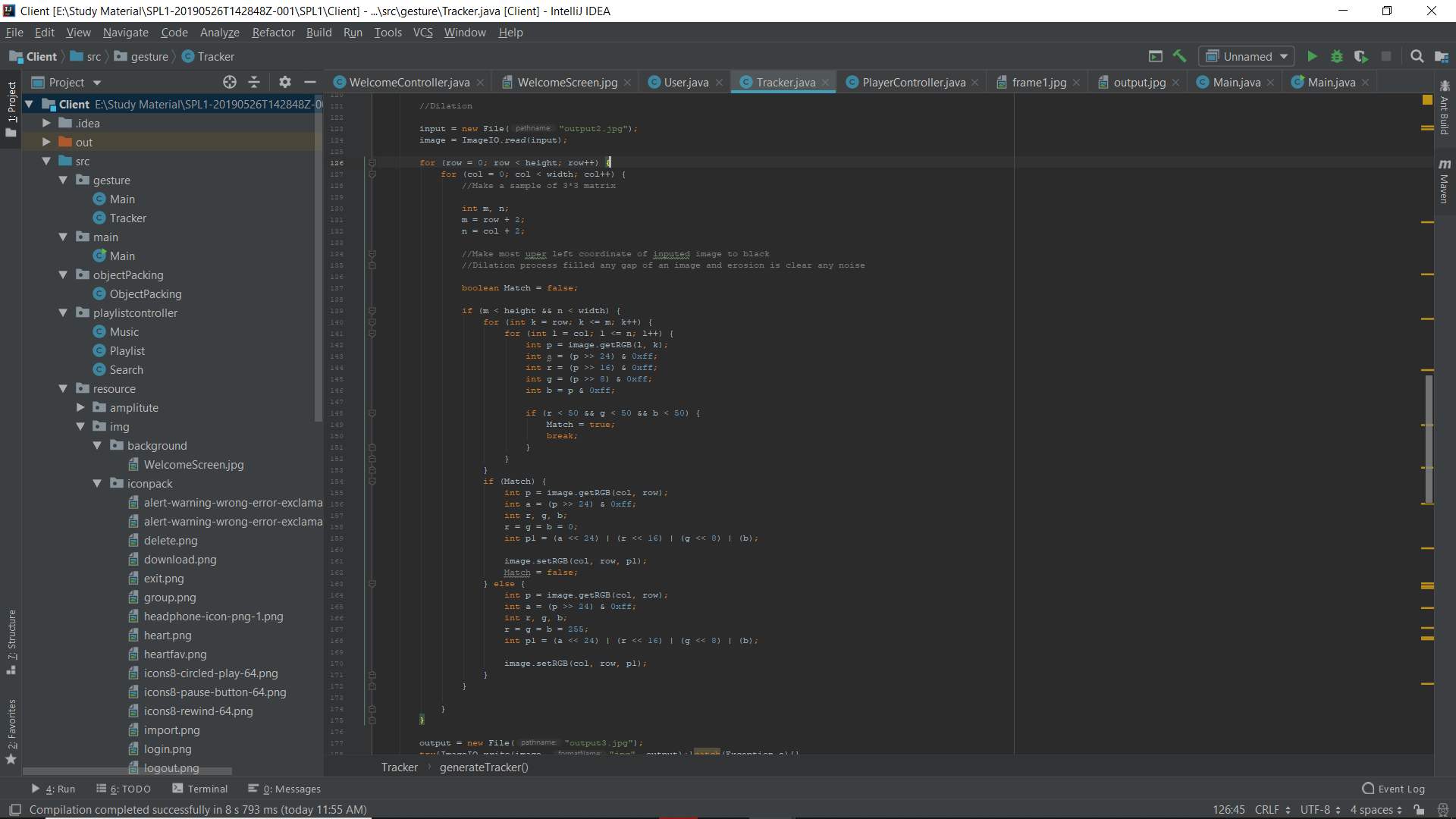


Figure (d)

Here mention that in line 142 in figure (d) we use getRGB() method to get color of a pixel. In java getRGB() is a built in method of BufferdImage class which is from java.awt.image package.

**getRGB**

public int getRGB()

Returns the RGB value representing the color in the default sRGB [ColorModel](https://docs.oracle.com/javase/7/docs/api/java/awt/image/ColorModel.html" \o "class in java.awt.image). (Bits 24-31 are alpha, 16-23 are red, 8-15 are green, 0-7 are blue).

**Returns:**

the RGB value of the color in the default sRGB ColorModel

.

**Since:**

JDK1.0

**1.1.2 Message Encryption Decryption**

**WAV file specifications**

**What's a WAV (RIFF) File?**  
A WAV (RIFF) file is a multi-format file that contains a header and data. For the purposes of this document, only a simple PCM file will be explored. A WAV file contains a header and the raw data, in time format.

**What's bit size?**  
Bit size determines how much information can be stored in a file.

**What is Sample Rate?**  
Sample rate is the number of samples per second. CD-Audio has a sample rate of 44,100. This means that 1 second of audio has 44,100 samples. DAT tapes have a sample rate of 48,000.

**What are Channels?**  
Channels are the number of separate recording elements in the data. For a real quick example, one channel is mono and two channels are stereo. In this document, both single and dual channel recordings will be discussed.

**What is the data?**  
The data is the individual samples. An individual sample is the bit size times the number of channels. For example, a monaural (single channel), eight bit recording has an individual sample size of 8 bits. A monaural sixteen-bit recording has an individual sample size of 16 bits. A stereo sixteen-bit recording has an individual sample size of 32 bits.

Samples are placed end-to-end to form the data. So, for example, if you have four samples (s1, s2, s3, s4) then the data would look like: s1s2s3s4.

**What is the header?**  
The header is the beginning of a WAV (RIFF) file. The header is used to provide specifications on the file type, sample rate, sample size and bit size of the file, as well as its overall length.

The header of a WAV (RIFF) file is 44 bytes long and has the following format:

|  |  |  |
| --- | --- | --- |
| **Positions** | **Sample Value** | **Description** |
| 1 - 4 | "RIFF" | Marks the file as a riff file. Characters are each 1 byte long. |
| 5 - 8 | File size (integer) | Size of the overall file - 8 bytes, in bytes (32-bit integer). Typically, you'd fill this in after creation. |
| 9 -12 | "WAVE" | File Type Header. For our purposes, it always equals "WAVE". |
| 13-16 | "fmt " | Format chunk marker. Includes trailing null |
| 17-20 | 16 | Length of format data as listed above |
| 21-22 | 1 | Type of format (1 is PCM) - 2 byte integer |
| 23-24 | 2 | Number of Channels - 2 byte integer |
| 25-28 | 44100 | Sample Rate - 32 byte integer. Common values are 44100 (CD), 48000 (DAT). Sample Rate = Number of Samples per second, or Hertz. |
| 29-32 | 176400 | (Sample Rate \* BitsPerSample \* Channels) / 8. |
| 33-34 | 4 | (BitsPerSample \* Channels) / 8.1 - 8 bit mono2 - 8 bit stereo/16 bit mono4 - 16 bit stereo |
| 35-36 | 16 | Bits per sample |
| 37-40 | "data" | "data" chunk header. Marks the beginning of the data section. |
| 41-44 | File size (data) | Size of the data section. |
| Sample values are given above for a 16-bit stereo source. | | |

So, that's the header. It shouldn't be difficult to write an application that creates the header

* + 1. **SHA-1**

In cryptography, SHA-1 is a cryptographic hash function designed by the National Security Agency (NSA) and published by the NIST as a U.S. Federal Information Processing Standard.

SHA stands for Secure Hash Algorithm.

The three SHA algorithms are structured differently and are distinguished as SHA-0, SHA-1, and SHA-2.

SHA-1 is very similar to SHA-0, but corrects an error in the original SHA hash specification that led to significant weaknesses. The SHA-0 algorithm was not adopted by many applications.

SHA-2 on the other hand significantly differs from the SHA-1 hash function. SHA-1 is the most widely used of the existing SHA hash functions, and is employed in several widely-used security applications and protocols.

In 2005, security flaws were identified in SHA-1, namely that a mathematical weakness might exist, indicating that a stronger hash function would be desirable. Although no successful attacks have yet been reported on the SHA-2 variants, they are algorithmically similar to SHA-1 and so efforts are underway to develop improved alternatives.

A new hash standard, SHA-3, is currently under development an ongoing NIST hash function competition is scheduled to end with the selection of a winning function in 2012.

**The SHA-1 hash function**

SHA-1 produces a 160-bit message digest based on principles similar to those used by Ronald L. Rivest of MIT in the design of the MD4 and MD5 message digest algorithms, but has a more conservative design.

The original specification of the algorithm was published in 1993 as the Secure Hash Standard, FIPS PUB 180, by US government standards agency NIST (National Institute of Standards and Technology). This version is now often referred to as SHA-0.

It was withdrawn by NSA shortly after publication and was superseded by the revised version, published in 1995 in FIPS PUB 180-1 and commonly referred to as SHA-1. SHA-1 differs from SHA-0 only by a single bitwise rotation in the message schedule of its compression function; this was done, according to NSA, to correct a flaw in the original algorithm which reduced its cryptographic security. However, NSA did not provide any further explanation or identify the flaw that was corrected. Weaknesses have subsequently been reported in both SHA and SHA-1. SHA-1 appears to provide greater resistance to attacks, supporting the NSA's assertion that the change increased the security.

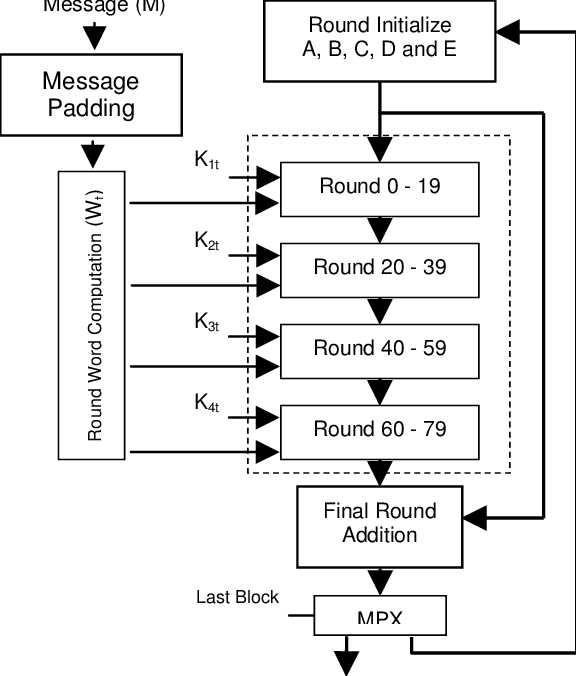


Figure (a)

**Implementation of SHA-1**

Figure (a) shows that first we need to take an arbitrary length of string which represent message of size M.

In the next step, we have to separate each character of the inputted string and convert each character to its corresponding ASCII value and put them into an array.

Now make an array of size L. Make sure that ‘L’ is a product of 512. And then convert the ASCII to binary value from inputted string and put it into this array and fill the empty portion with ‘0’.

Figure (b) shows an implementation of an above steps.

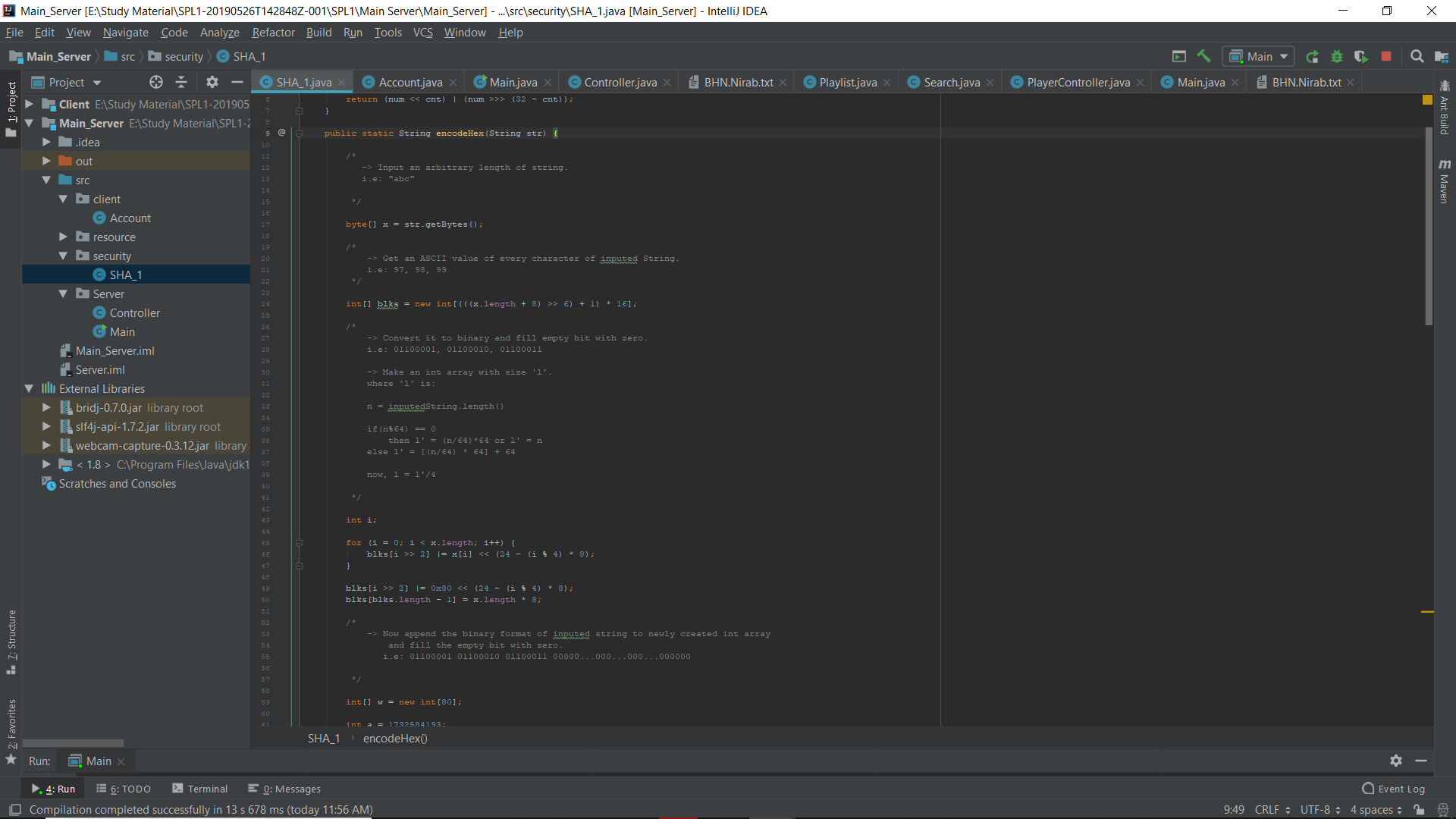


Figure (b)

In the next steps create an array of size 80 and initialize 5 constant.

-> create an int array of size 80 and initialize 5 constant -  
  
(a) H\_0 = 1732584193  
(b) H\_1 = -271733879  
(c) H\_2 = -1732584194  
(d) H\_3 = 271733878  
(e) H\_4 = -1009589776

Figure (c)

Now manipulate those 5 constant with some iteration or round to get a 160 bit message digest.

For manipulating we use left rotation. For left rotation we use :

w[i] = (w[i-3] **xor** w[i-8] **xor** w[i-14] **xor** w[i-16])

and finally we get our 160 bit message digest.

int[] words = {a, b, c, d, e};  
StringBuilder sb = new StringBuilder();  
  
for (int word : words) {  
 String hexWord = Integer.*toHexString*(word);  
 while (hexWord.length() < 8) {  
 hexWord = "0" + hexWord;  
 }  
 sb.append(hexWord);  
}  
  
/\*  
 -> After manipulate those constant H\_0 ... H\_4  
 -> Now take a Hexadecimal representation of them.  
 i.e: H\_0 = -1449574858  
 Hex(H\_0) = a9993e36  
  
 \*/  
  
return sb.toString();  
  
/\*  
 -> The output of message digest is always 160 bits/20bytes/5 int size.  
 -> The total character of message digest is 40. Because each Hexa representation of  
 H\_0...H\_4 is 8 bits and total bits is 160.  
  
 \*/

Figure (d)

**1.2 Challenge**

The challenge I faced that how I can detect the movement of gesture and perform a task? I have to overcome it. The idea behind it that, first I took an image using webcam. Then I am trying to find a region where RED color exist in each frame. I track that portion of RED color of an image and apply thresholding effect on an images. Where every pixel without RED are WHITE and RED pixel of an image are BLACK.

Then I do morphological operation on that frame and apply erosion and dilation so that I am able to find a perfect RED portion of each frame.

After taking certain number of frame I am storing position of (X,Y) in an array so that in future I am able to make a decision using this array.

For example I have an array which is filled by the tracker function. Where tracker function gives a position of (X,Y) where RED color exits of each frame after doing some processing.

|  |
| --- |
| **Position pos[5] = { (4,5), (7,5), (10,5), (13,5), (17,6) }** |

In this example, we consider a duration of 5 frames. Where each frame gives a location if any RED color exits on that frame. Now I can make a decision from movement of a gesture and perform a task.

If we see ‘pos’ array then we observe that ‘X’ is increasing and the change of ‘Y’ is less. So, we can conclude that the movement is from left to right and if the value of ‘X’ is decreasing and the change of ‘Y’ is less then we can say that the movement of gesture is from right to left. On the other contrary if the value of ‘Y’ is increasing and the change of a ‘X’ is less then we can say that the movement is from bottom to top. And if the value of ‘Y’ is decreasing and the change of ‘X’ is less then we can conclude that the movement is from top to bottom.

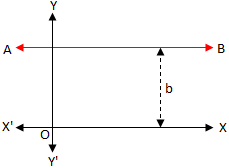
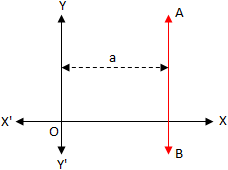
 

Figure (a)

**Implementation of challenging portion**

Tracker track = new Tracker();

try{

track.generateTracker("frame1.jpg", 315, 360, 55, 55);

}

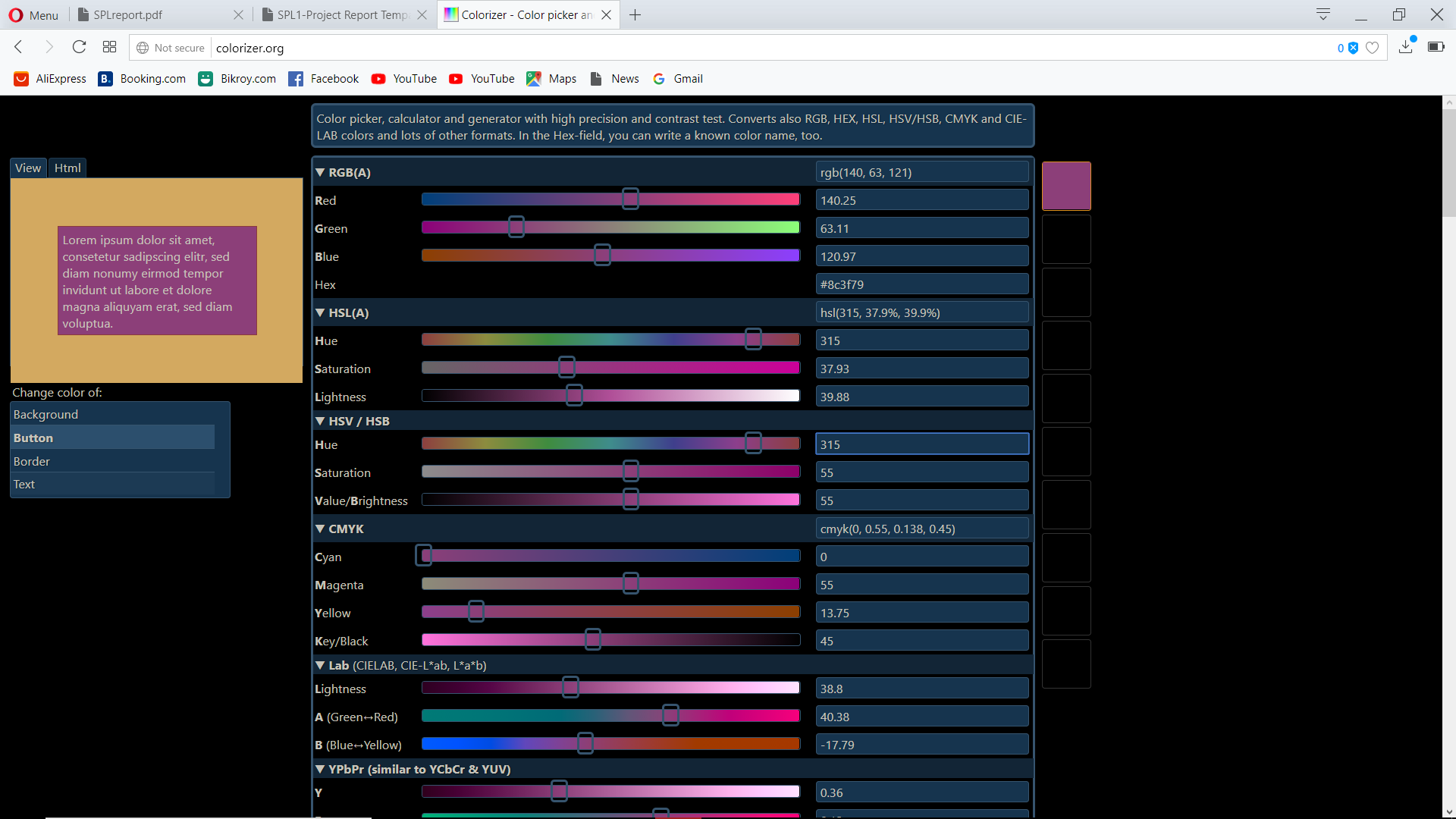
catch(Exception e){}

int pos[] = track.getPos();

In this **pos** array **track.getPos()** method return (X,Y) position of a frame where RED color exists. Pos[0] is the value of ‘X’ and pos[1] = value of ‘Y’ . If a frame has no RED color then **track.getPos()** return [-1, -1].

**track.generateTracker("frame1.jpg", 315, 360, 55, 55)**

In this line track.generateTracker() method take 5 parameters. The first parameter is for captured image which is captured by webcam. And the rest of the parameters represent the range of a HUE and Saturation of a RED color.



The reason of taking HUE and Saturation is for I use HSB (HUE, SATURATION and BRIGHTNESS/VALUE)colorspace to find the region of RED color in each frame.

float[] hsb = Color.*RGBtoHSB*(r, g, b, null);  
  
if (

(hsb[0]\*360 >= lowerBoundHue && hsb[0]\*360 <= upperBoundHue)

&&

(hsb[1]\*100 >= lowerBoundSaturation) && (hsb[2]\*100 >= lowerBoundValue)

)

r1 = g1 = b1 = 0;

else   
 r1 = g1 = b1 = 255;

**Color.RGBtoHSB()** method is a built in method of java where it gives a HSB colorspaces from RGB colorspaces. The first index of hsb array represent the Hue and the second index of hsb array represent saturation and last one represent Brightness value and it gives value from 0 to 1 so that for calculating Hue we have to multiply it with 360 and for calculating Saturation we have to multiply it with 100.

And finally, we find the region of RED color from given frame.

After capturing 7-10 frame I’m storing the position and finally I’m able to detect motion. If user swipe from left to right then next song will be play and if user swipe from right to left then previous song will be play. Similarly if an user swipe from top to bottom Volume of the song will be decreasing and swiping bottom to top Volume of the song is increasing. And the mechanism of this portion has been discussed already.

1. **Project Overview**

**2.1** Import multiple song at a time and make a playlist

**2.2** Play song from playlist

**2.2.1** Manually

**2.2.2** Using Gesture

**2.3** Added song into favorite list

**2.3.1** Manually set a rating out of 5

**2.3.2** Automatically detect whether a song is liked by the user or not and add it to favorite list

**2.3.3** Automatically track user favorite list and remove song from favorite list if user doesn’t listen it certain period of time

**2.4** User account

**2.4.1** Create account [ Password is encrypted using SHA-1]

**2.4.2** Login

**2.5** Server

Handle user account and playlist

**2.6** Message Encryption and Decryption

**2.6.1** Encryption

**2.6.2** Decryption

**2.0 Scan the QR code to watch the full features of MUSIC HUNTER**



<http://bit.ly/2JJ2vGu>

**2.1 Import multiple song at a time and make a playlist**

User can import multiple song at a time and make a playlist. And this playlist is safely store in the server so that there is no chance to lose user playlist.

public void addMusictoPlaylist(String userID, File path, int rating);

Here **addMusictoPlaylist()** method takes three parameters where firstone is for userID. So that, SERVER can recognize for whom to store the playlist and second one is for file path of a song and last one is for rating.

**2.2 Play song from playlist**

**2.2.1 Manually**

This portion of code in figure (a) is for manually playing a song from playlist. When a user press a play button an ActionEvent are occurs. Then it show a pane with animation where music is playing and that pane is holding some information (meta data) of a song.

Then it load song from playlist and start playing it.

public void setPlayPauseButton(ActionEvent event) {  
  
 ObservableList<Label> btn = playlistHolder.getSelectionModel().getSelectedItems();  
  
 Label tmp = null;  
 for (Label b : btn) tmp = b;  
  
 fileName = tmp.getText();  
  
 char ch[] = new char[fileName.length()];  
 ch = fileName.toCharArray();  
 fileName = "";  
 for (int i = 2; i < ch.length; i++) fileName += (ch[i] + "");  
  
 Scanner testread = null;  
 try {  
 testread = new Scanner(new File("src/resource/playlist/" + WelcomeController.*currUser* + ".txt"));  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }

while (testread.hasNextLine()) {  
 String info = testread.nextLine();  
 String sub[] = info.split("\\?");  
 Music tmp1 = new Music();  
 tmp1.path = new File(sub[0]);  
 tmp1.rating = Integer.*valueOf*(sub[1]);  
 tmp1.priority = Integer.*valueOf*(sub[2]);  
 tmp1.dateToAddedFavList = sub[3];  
  
 if (tmp1.path.getName().equals(fileName)) {  
  
 currMusic = new Music();  
 currMusic.path = tmp1.path;  
 currMusic.rating = tmp1.rating;  
 currMusic.priority = tmp1.priority;  
 currMusic.dateToAddedFavList = tmp1.dateToAddedFavList;  
 media = new Media(tmp1.path.toURI().toString());  
 mediaPlayer = new MediaPlayer(media);  
 mediaview.setMediaPlayer(mediaPlayer);  
 mediaPlayer.play();  
 break;  
  
 }  
 }  
  
  
 } catch (Exception e) {  
 System.*out*.println(e);  
 }  
  
  
 *is\_play* = true;  
 visualizer();  
 openInfo();  
  
 } else {  
 FadeTransition ftVisualizerHolder = new FadeTransition(Duration.*millis*(2000), visualizerHolder);  
 ftVisualizerHolder.setFromValue(0.7);  
 ftVisualizerHolder.setToValue(0);  
 ftVisualizerHolder.play();  
 playPauseImage.setImage(new Image("/resource/img/iconpack/icons8-circled-play-64.png"));  
 if (mediaPlayer != null)  
 mediaPlayer.pause();  
  
  
 *is\_play* = false;  
 }  
}

Figure (a)

**2.2.2 Using Gesture**

User can play a song using gesture. When a user swiping a RED color object in front of the webcam then MUSIC HUNTER can recognize the movement of a user and perform a task.

How MUSIC HUNTER can perform a task that was described in **1.2**

**2.3 Added song into favorite list**

**2.3.1 Manually**

User can set a rating of a song and song will be added into favorite list. The rating is from 0 to 5. Where 0 is a default rating. In case of default rating a song is not consider as a favorite. Otherwise it is consider as a favorite song and added into favorite list.

In figure (a) showing that how a **setRating()** method works. This method takes three parameters where first one is for userID and second one is for targeted song which I want to set rating and the last one is for user rating.

After giving those parameter **setRating()** my program willsearch the targeted song from database and set rating into it and after setting the rate of a song it will saved the song to database.

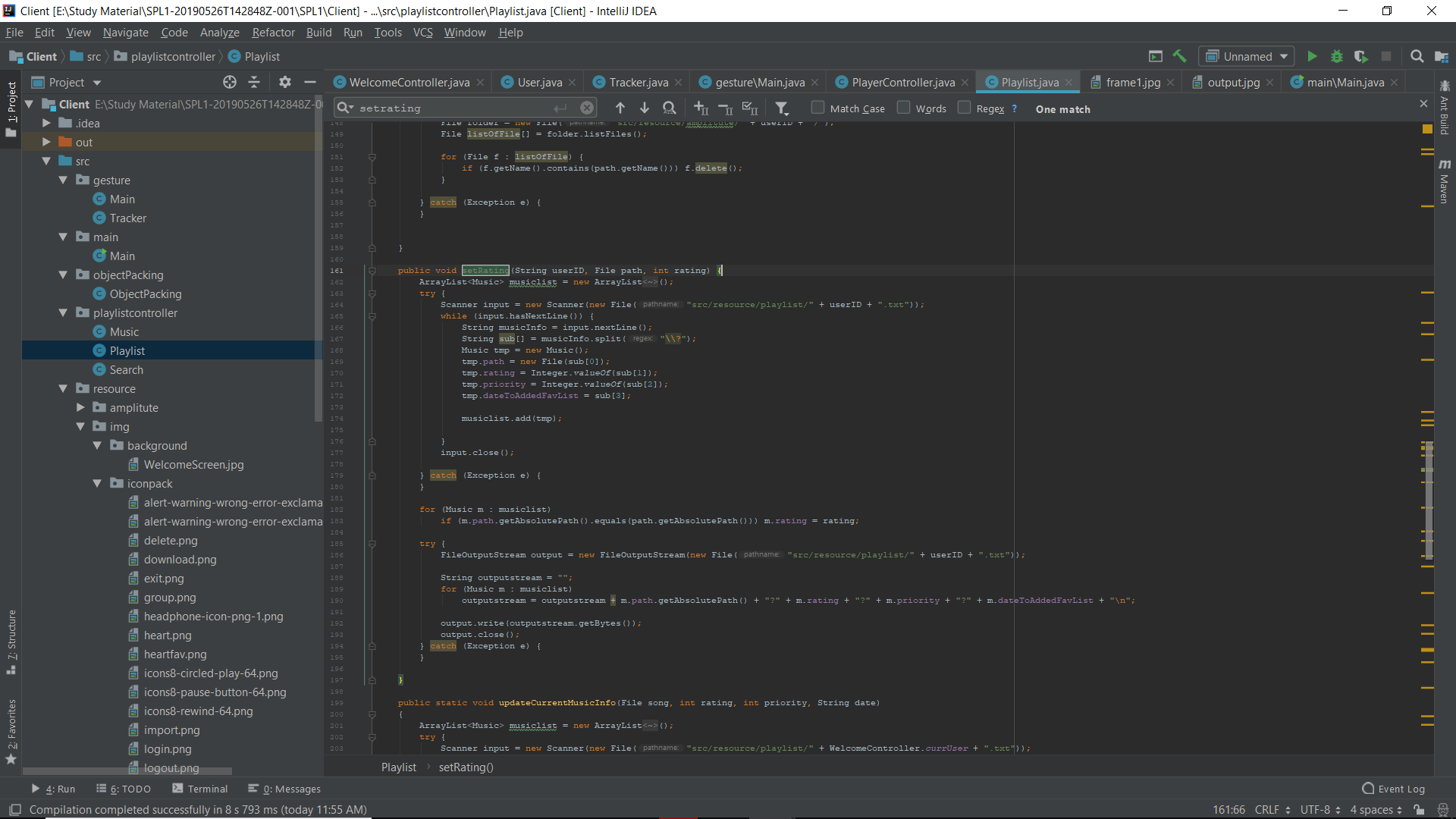


Figure (a)

And this is how manually I can set a rating of a song.

**2.3.2 Automatically detect whether a song is liked by the user or not and add it to favorite list**

MUSIC HUNTER are continuously track user state. If an user listen a song frequently then it might be a favorite song for an user. Sometimes user is forget to set a rating of a song. But don’t worry. The algorithm of MUSIC HUNTER has taken the responsibility of it.

If a user listened a song 70% of total duration without seeking a song and without set a manual rating then MUSIC HUNTER will assume that user listened it and priority of that song will increasing by one.

if (current.toSeconds() >= percent.get && !isSeekingOccurs && !isListened && !isClickRatingSlider && currMusic.priority <= 10) {

currMusic.priority++;

Playlist.*updateCurrentMusicInfo*(currMusic.path, currMusic.rating, currMusic.priority, currMusic.dateToAddedFavList);

updatetmpMusic();  
 isListened = true;

Figure (a)

Figure (a) showing that how priority is increasing for a particular song.

int tmppercent = (int)((mediaPlayer.getTotalDuration().toSeconds()\*70)/100);  
percent.set(tmppercent);

This is how I calculating the percentage of total duration.

if (currMusic.priority == 0) {

String formatedDate = "01-Jan-2000 23:01:01";  
 currMusic.rating = 0;  
 currMusic.dateToAddedFavList = formatedDate;

Playlist.*updateCurrentMusicInfo*(currMusic.path, 0, currMusic.priority, formatedDate);

updatetmpMusic();

} else if (currMusic.priority == 2) {

Date dateAddedtoFavlist = new Date();  
  
 SimpleDateFormat format = new SimpleDateFormat("dd-MMM-yyyy HH:mm:ss");  
 String formatedDate = format.format(dateAddedtoFavlist);  
  
 currMusic.rating = 1;  
 currMusic.dateToAddedFavList = formatedDate;

Playlist.*updateCurrentMusicInfo*(currMusic.path, 1, currMusic.priority, formatedDate);

updatetmpMusic();

}

// and so on

Figure (b)

In figure (b) I am showing that if priority is 0 then rating of a song has to be default value 0 and set the **dateToAddedFavList** attribute to default value **"01-Jan-2000 23:01:01"** which represent this song has no rating.

And if the priority is 2 or grater then my program captured current date and store it to **dateToAddedFavList** attribute which represent when a song is added into favorite list.

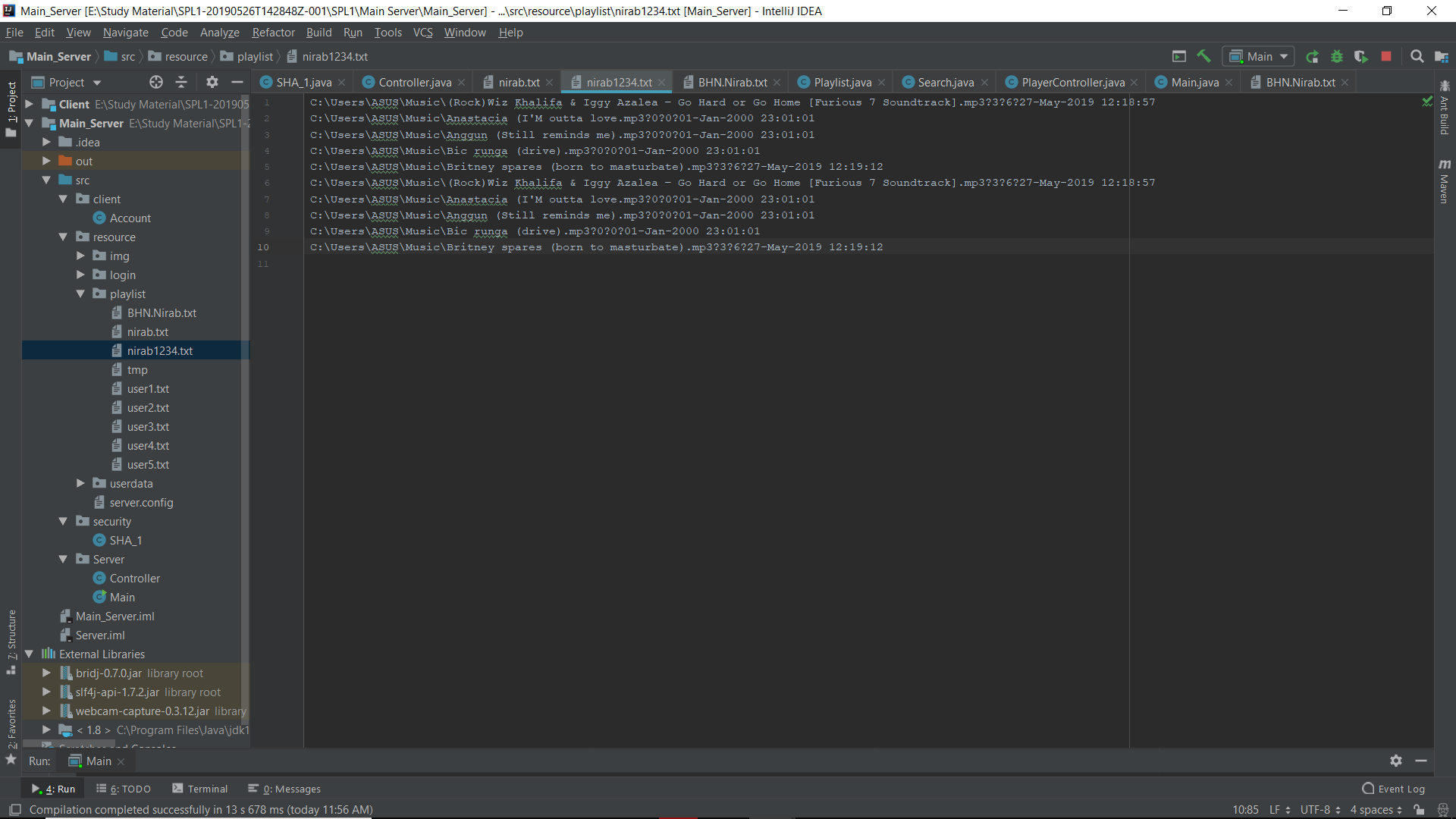


Figure (c)

C:\Users\ASUS\Music\Britney spares (born to masturbate).mp3?3?6?27-May-2019 12:19:12

Figure (c) showing the format of an user playlist. Where ‘?’ is used for delimiter and first portion before the ‘?’ mark represent the path of a music and second one is for rating and third one is for priority of a song and last one is for when a music was added into favorite list.

And this is a mechanism to added music into favorite list automatically by tracking an user state.

**2.3.3 Automatically remove song from favorite list if user doesn’t listen it certain period of time**

When an user liked a song he/she will listen it thousand times. After listening it thousand time he/she will start ignoring it. But it will still remain in favorite list. But don’t worry. MUSIC HUNTER has taken the responsibility to remove unwanted song from user favorite list.

long duration = currDate.getTime() - addedDate.getTime();  
  
  
if(tmp1.rating>0)  
{  
 if(TimeUnit.*MILLISECONDS*.toSeconds(duration)>=10)  
 {  
 tmp1.rating--;  
 tmp1.priority = tmp1.priority-2;  
 }  
}

Figure (a)

To remove music from favorite list automatically, I calculate the duration by subtracting current date and date added into favorite list.

If an user doesn’t listened a song certain period of time then the priority of a song is decreasing by 2 and rating of a song is also decreasing by one and so on. If a rating will be 0 then the song is automatically removed from user favorite list.

**2.4 User Account**

**2.4.1 Create account [ Password is encrypted using SHA-1]**

User can create an account along with a private playlist. For creating an account user have to give first name, last name, userID and password. User can also set a profile picture. If user doesn’t provide a profile picture then a default profile picture will be set for an user.

It is mentioned that password of an user will be encrypted using SHA-1 algorithm and store it safely to the server. If user doesn’t provide first name or last name or userID or password he/she will not be able to create an account. User will also not be able to create an account if he/she will provide a duplicate userID.

**2.4.2 Login**

User can login into MUSIC HUNTER using his/her userID and password. When user give those required input the input will go to the server and server gives a validity to the user to login. After login, my program will maintain a session to keep the track of an user.

**2.5 SERVER**

Server is for handle user account and user playlist. In this section I’ll discuss how my sever works.

First it take a request from client and run a thread for each client.

ServerSocket server = new ServerSocket(PORT);  
  
while(true)  
{  
 System.*out*.println("Waiting for connection...");  
 Socket client = server.accept();  
 System.*out*.println("Connected with: " + client.toString() );  
 Thread eachClient = new Controller(client);  
 eachClient.start();  
  
}

The mechanism to serve a client is discuss below. To send and receive data, server and client use a class called **ObjectPacking**.

public class ObjectPacking implements Serializable {  
  
 public String command;  
 public String from;  
 public String to;  
 public String comment;  
  
 public Object data;  
  
}

figure (a)

In figure (a) shows how **ObjectPacking** class are made. This is one kind of packet where “command” attribute represent which task have to perform in the server. If “command” is “createAccount” then server is preparing to create an account and so on.

“from” and “to” attribute represent from and destination and if user have any comment about packet he/she can put that comment in “comment” attribute. It is optional. And finally the “data” attribute hold the actual data.

Client or Server make an object of **ObjectPacking** class and send it throw the socket.

if(object.command.equals("createAccount"))  
{

User user = (User)object.data;  
 Account acc = new Account();  
 Boolean feedback = acc.createAccount(user);  
  
 response.command = "createAccount";  
 response.from = "server";  
 response.to = user.userID;  
  
 if(feedback) response.comment = "true";  
 else response.comment = "false";  
  
}

Figure (b)

Here figure (b) shows how server is response to the client request. It read the “command” portion of **ObjectPacking** classobject and make a decision. After making decision it will give a response to a client.

**2.6 Message Encryption and Decryption**

**2.6.1 Encryption**

User can push a message into WAV file audio data and after pushing the message the audio holds some distortion. But the change of an audio is impossible to recognize.

To push message into a audio data first we need to declare some attribute to hold the header of WAV file. Figure (a) shows that how to make a space for each of the header portion.

public Encryption(File path)  
{  
 orginalFile = path;  
 chunkSize = new byte[4];  
 subchunkSize = new byte[4];  
 audioFormat = new byte[2];  
 numberOfChannels = new byte[2];  
 sampleRate = new byte[4];  
 byteRate = new byte[4];  
 blockAlign = new byte[2];  
 bitsPerSample = new byte[2];  
 dataSubchunkSize = new byte[4];  
  
 readOrginalFile();  
}

Figure (a)

Then **readOrginalFile()** method initialize those array with appropriate value. **readOrginalFile()** method is shown in figure (b).

private void readOrginalFile()  
{  
 DataInputStream input = null;  
  
 try{  
  
 input = new DataInputStream(new FileInputStream(orginalFile));  
  
 chunkIdentifier = "" + (char)input.readByte() + (char)input.readByte() + (char)input.readByte() + (char)input.readByte();  
 input.read(chunkSize);  
 format = "" + (char)input.readByte() + (char)input.readByte() + (char)input.readByte() + (char)input.readByte();  
  
 subchunkIdentifier = "" + (char)input.readByte() + (char)input.readByte() + (char)input.readByte() + (char)input.readByte();  
 input.read(subchunkSize);  
 input.read(audioFormat);  
 input.read(numberOfChannels);  
 input.read(sampleRate);  
 input.read(byteRate);  
 input.read(blockAlign);  
  
 input.read(bitsPerSample);  
 dataSubchunkIdentifier = "" + (char)input.readByte() + (char)input.readByte() + (char)input.readByte() + (char)input.readByte();  
 input.read(dataSubchunkSize);  
  
 soundData = new byte[(int)*byteArrayToLong*(dataSubchunkSize)];  
 input.read(soundData);  
  
 }catch(Exception e){}  
  
}

Figure (b)

Now its time to push message into audio data. In figure (c) shows how to push message into audio data. I push each character of a message after each 5000 sample of audio data.

public boolean injectData(String data)  
{  
 if(data.length() >= soundData.length/5000) return false;  
  
 for(int i=0,j=0; i<soundData.length && j<data.length(); i++)  
 {  
 if(i%5000==0){  
 int dataAtj = (int)data.charAt(j);  
 soundData[i] = Byte.*valueOf*(Integer.*toString*(dataAtj));  
 j++;  
 }  
 }  
 writeOrginalFile();  
 return true;  
}

Figure (c)

And finally I write the manipulated audio into file.

**2.6.2 Decryption**

For decrypt message I have to read audio which holds the actual message. Figure (a) show how to get encrypted messages from audio data.

public String getEncryptData()  
{  
 skippedHeader();  
  
 String encodedData = "";  
  
 for(int i=0; i<soundData.length; i++){  
 if(i%5000==0) {  
 int value = soundData[i];  
 if(value == 46) break;  
 encodedData = encodedData + (char)value;  
 }  
 }  
 return encodedData;  
  
}

Figure (a)

In figure (a) first I have to skip header because I don’t need header at this time. Then I travers the audio data and find the character that I included when I encrypt it after each 5000 sample.

The messages is ends with full stop (.) and if the value of audio sample is 46 then loop should be break and return expected string.

1. **User Manual**

**3.1 Create an account**

To create an account user should give first name, last name userID and password. Every field of

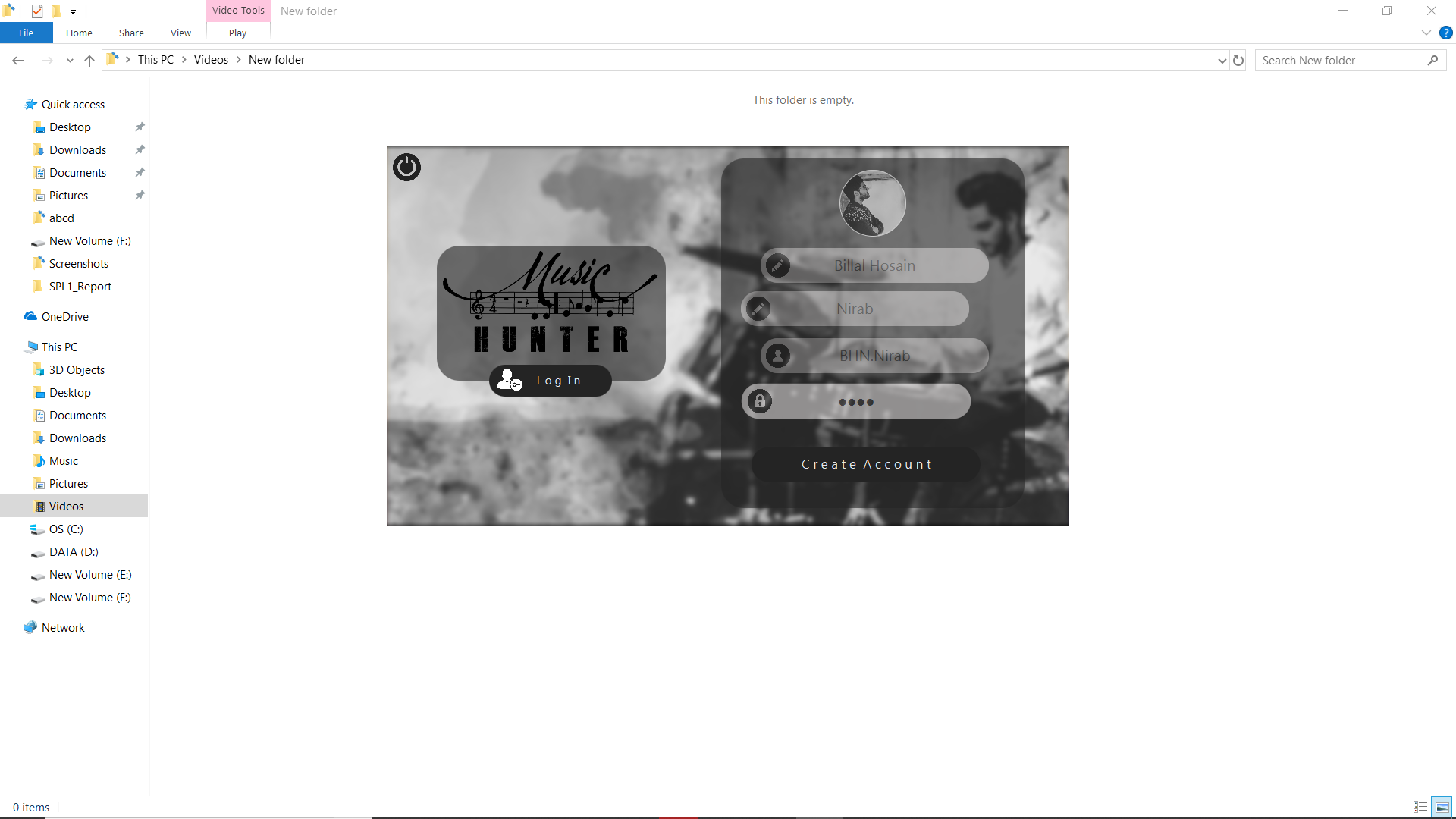


Figure (a)

this form is compulsory. It hasn’t option to skip any of those fields. User can set a profile picture and can skip it. If user doesn’t set any profile picture then default profile picture will be set.

In figure (a) I will show you how to create an account.

**3.2 Login**

In figure (a) I will show you how to login. To login into your account you must provide your userID and password.

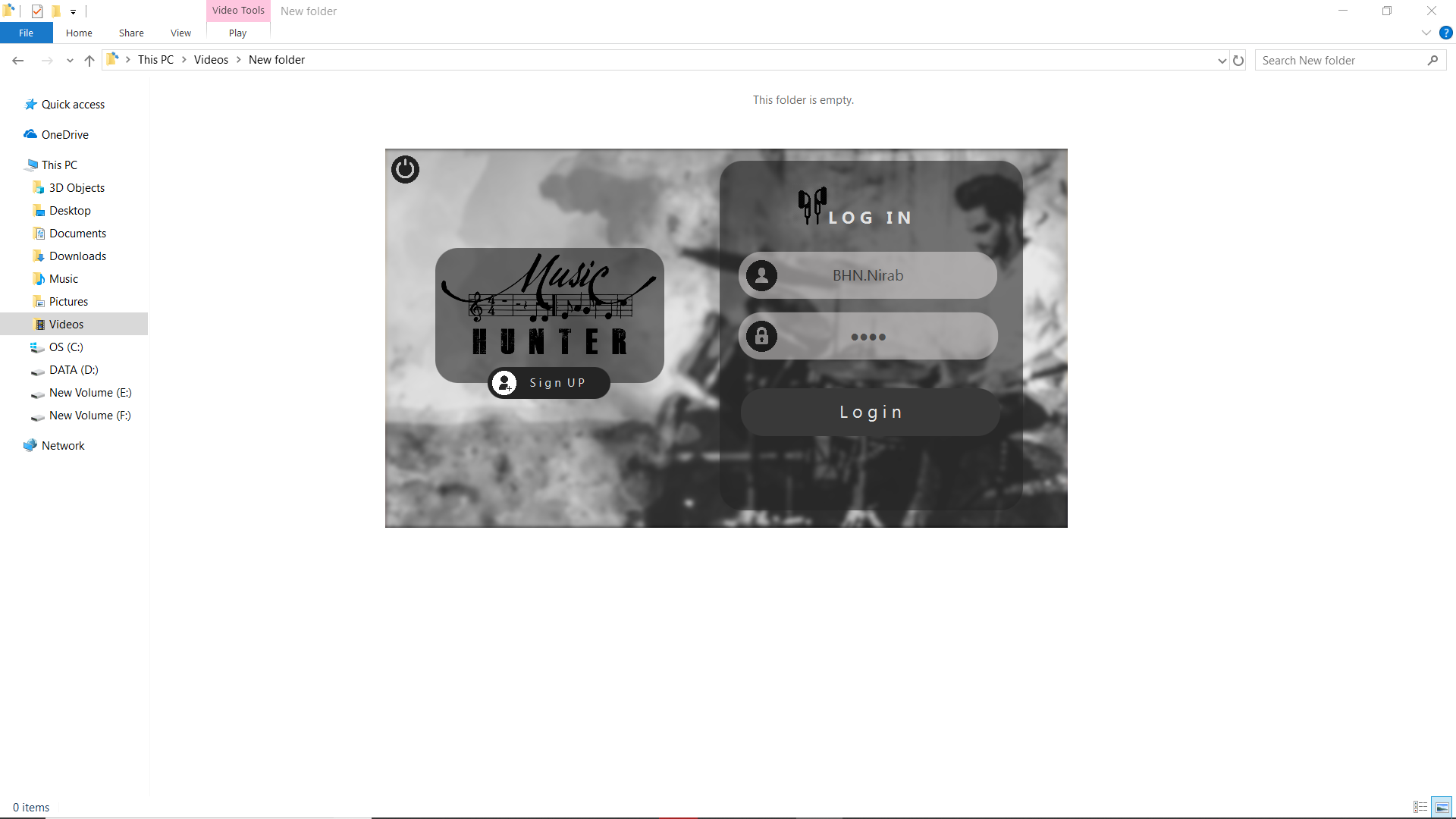


Figure (a)

**3.3 Import song into playlist**

In figure (a) shows that how to import song into playlist. To import song into playlist click the playlist icon from the left top side of an application and then playlist sidebar is open.

After opening playlist sidebar click the top right corner of an application and then file chooser will be open.

After opening file chooser select multiple song as you want and click open as shown in figure (b)

After importing all of song it will like figure (c).

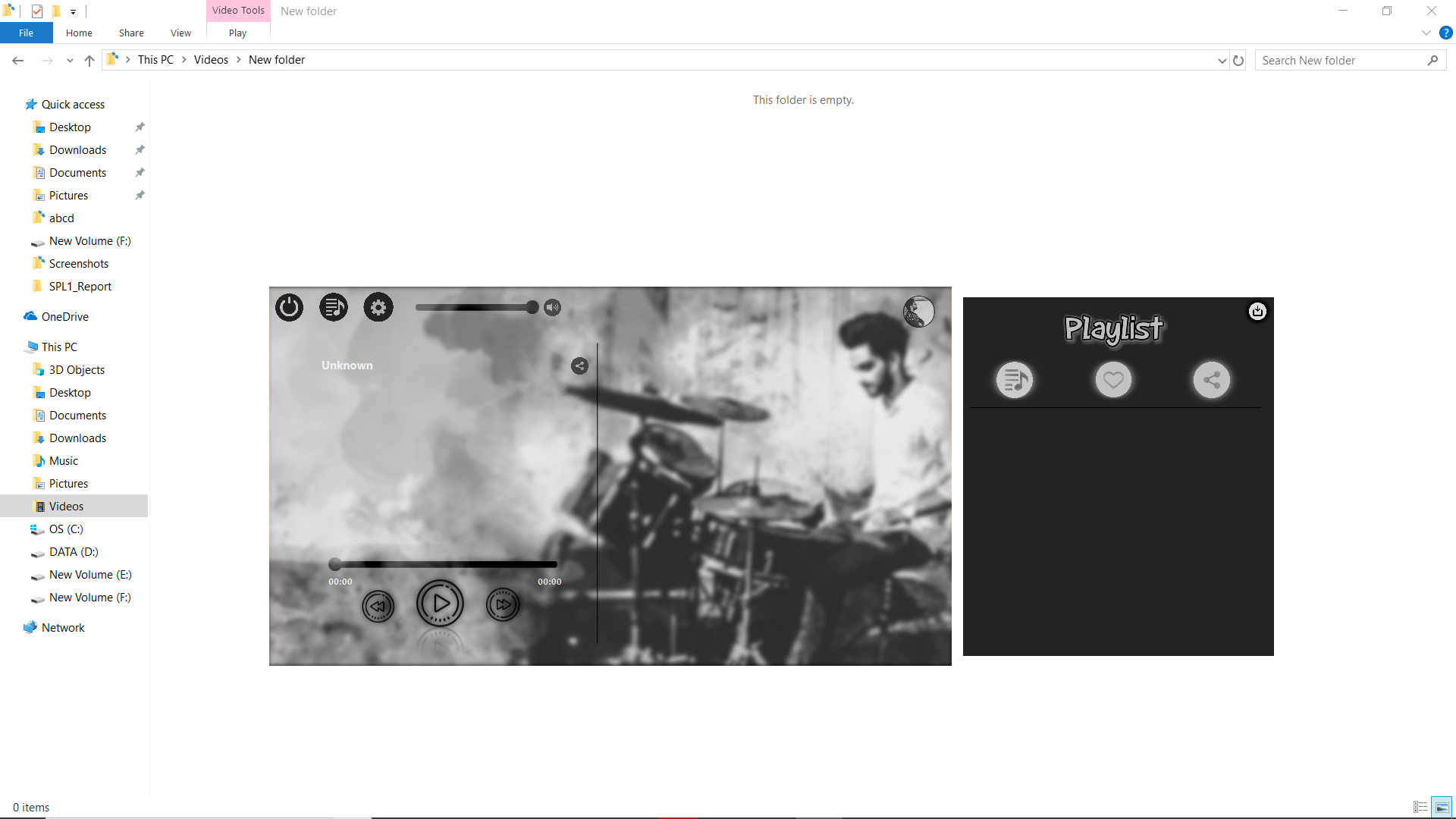


Figure (a)

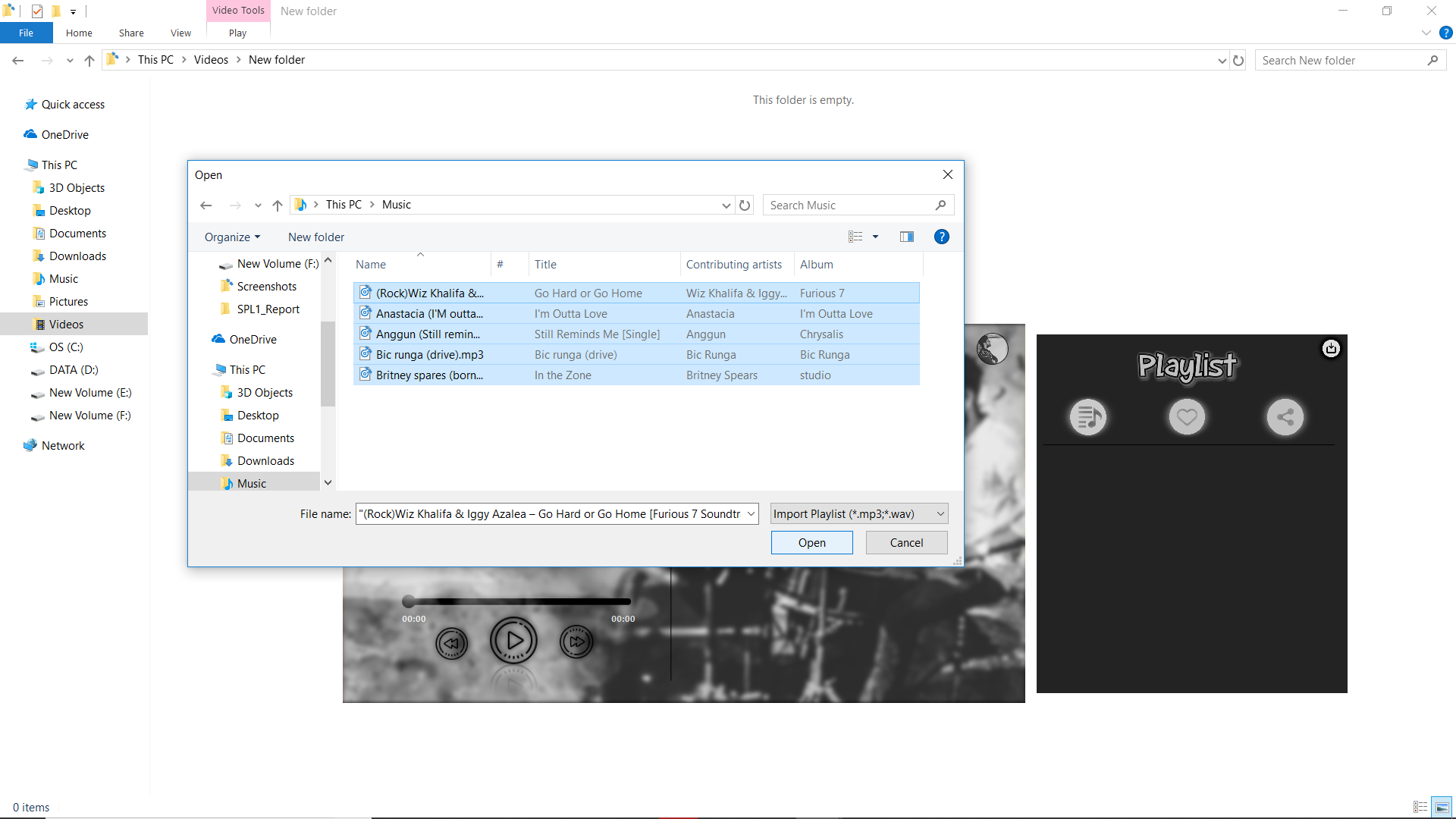


Figure (b)

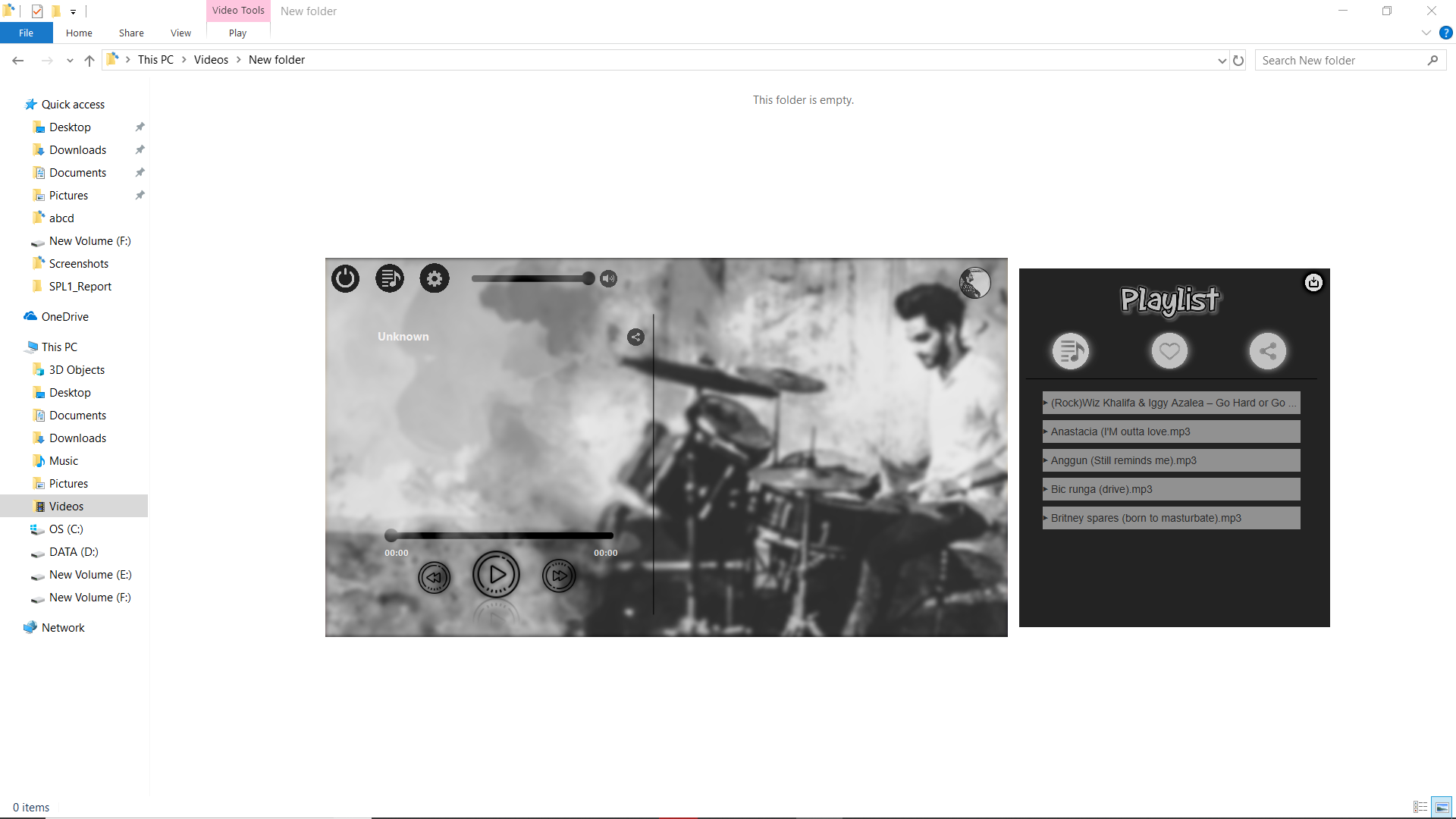


Figure (c)

**3.4 Play song from playlist**

**3.4.1 Manually**

To play song manually select song from play list and click play button shown in figure (a).

To play next song, click next button and playing previous song click previous button.

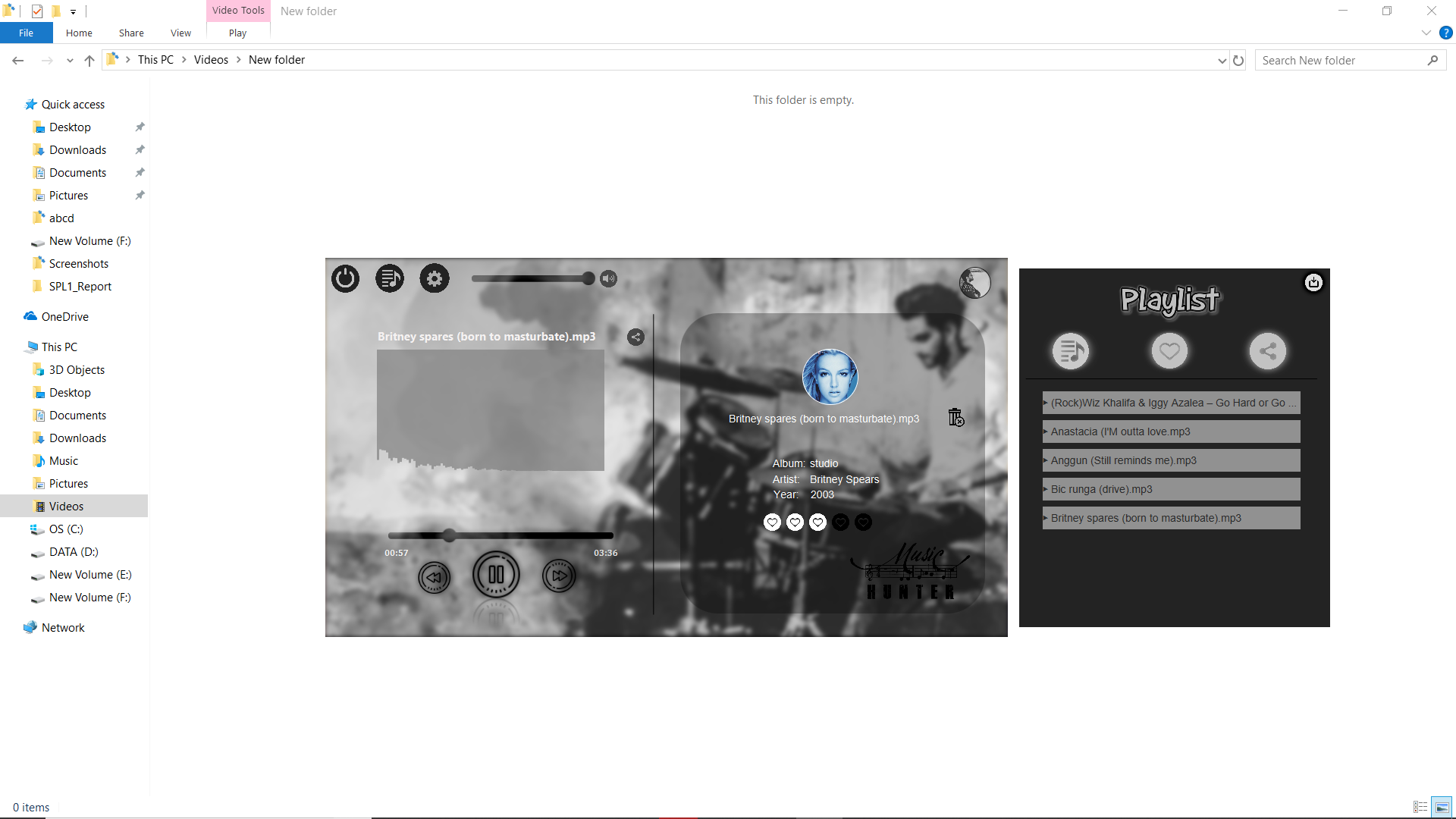


Figure (a)

To change volume using volume slider, slide the slider left and right using mouse shown in figure (b)

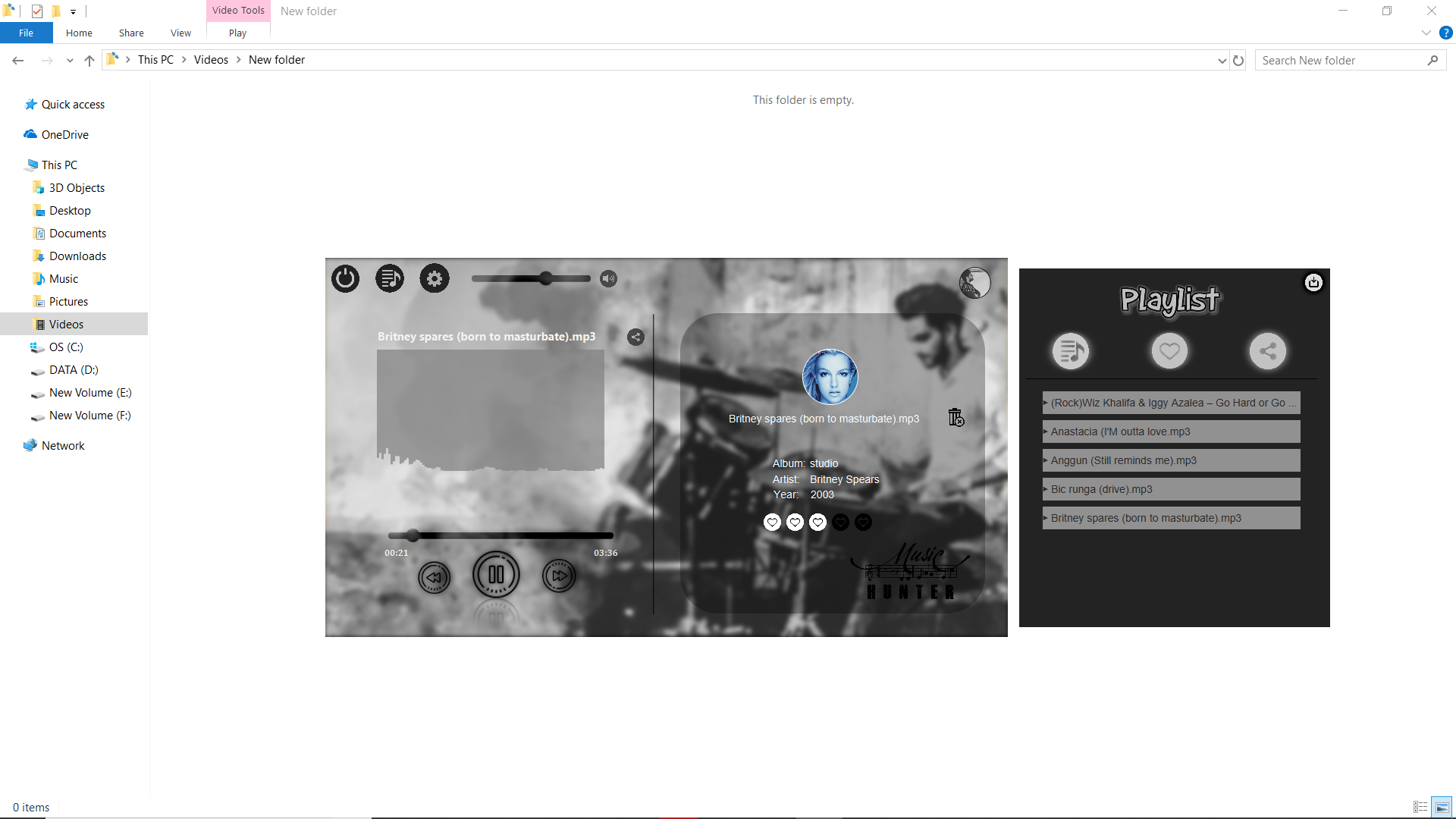


Figure (b)

User can also mute a song by clicking sound icon. Once a sound icon is clicked by the user the playing song will be muted and if user clicked sound icon again the song will be playing in full sound which is shown in figure (c). And user can seeking a song by sliding the timer.

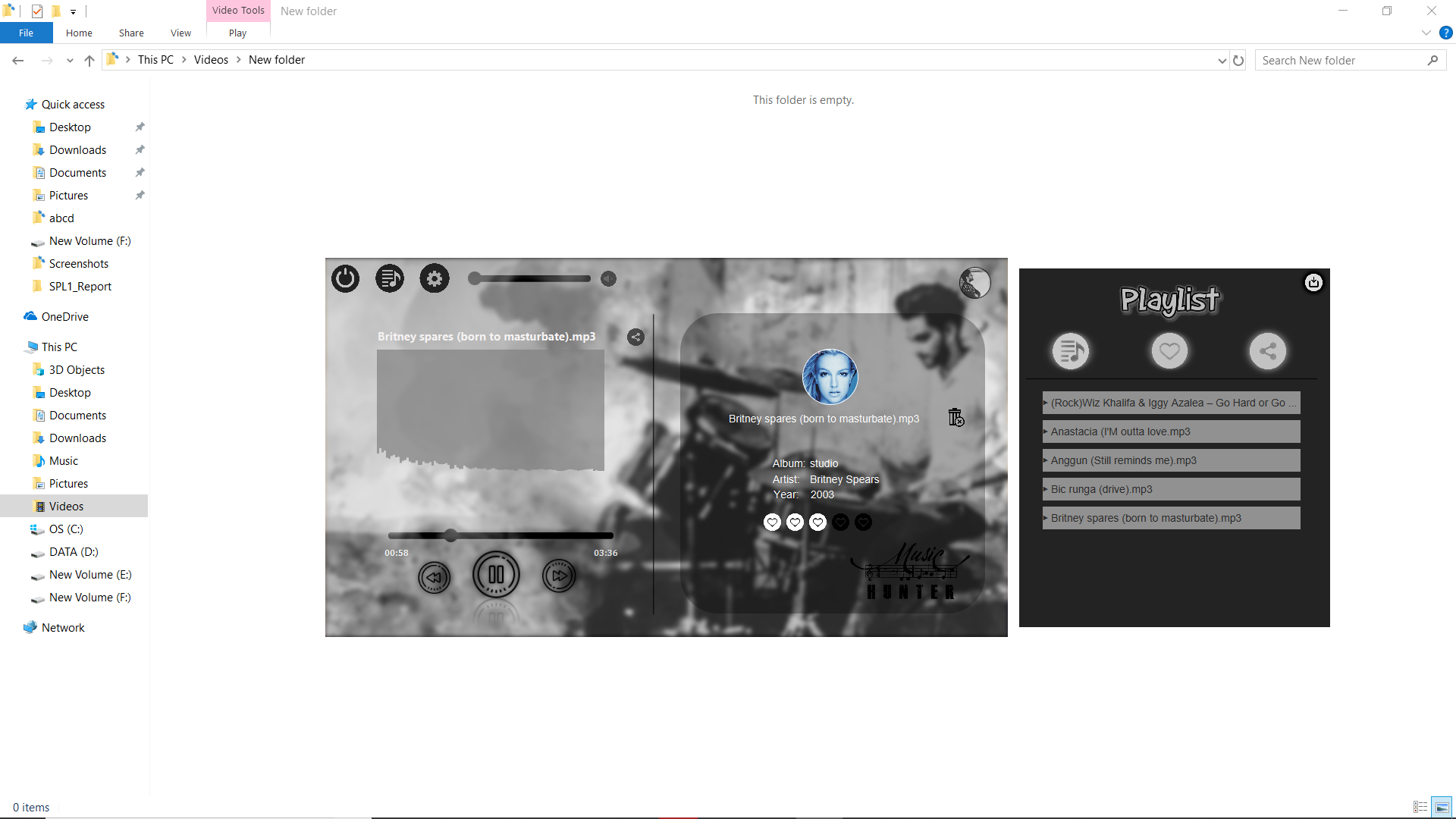


Figure (c)

**3.4.2 Using gesture**

There are 4 command to control music-

1. Swipe left to right to play next song.
2. Swipe right to left to play previous song.
3. Swipe bottom to top to increase volume.
4. Swipe top to bottom to decrease volume.

To use gesture first you have to pick a RED color object. It can be anything which is RED color or DARK RED color.



Figure (a)

Figure (a) show the sample color that you have to pick. Then using this object now you are able to control music using gesture. Put this object in front of the webcam and make a movement to give command.

**3.5 Added song into favorite list**

**3.5.1 Manually**

To set a rating of a song slide your mouse on star icon. In figure (a) there are 5 star. You have to option to give a rating of a song from 0 to 5 where 0 is a default value.

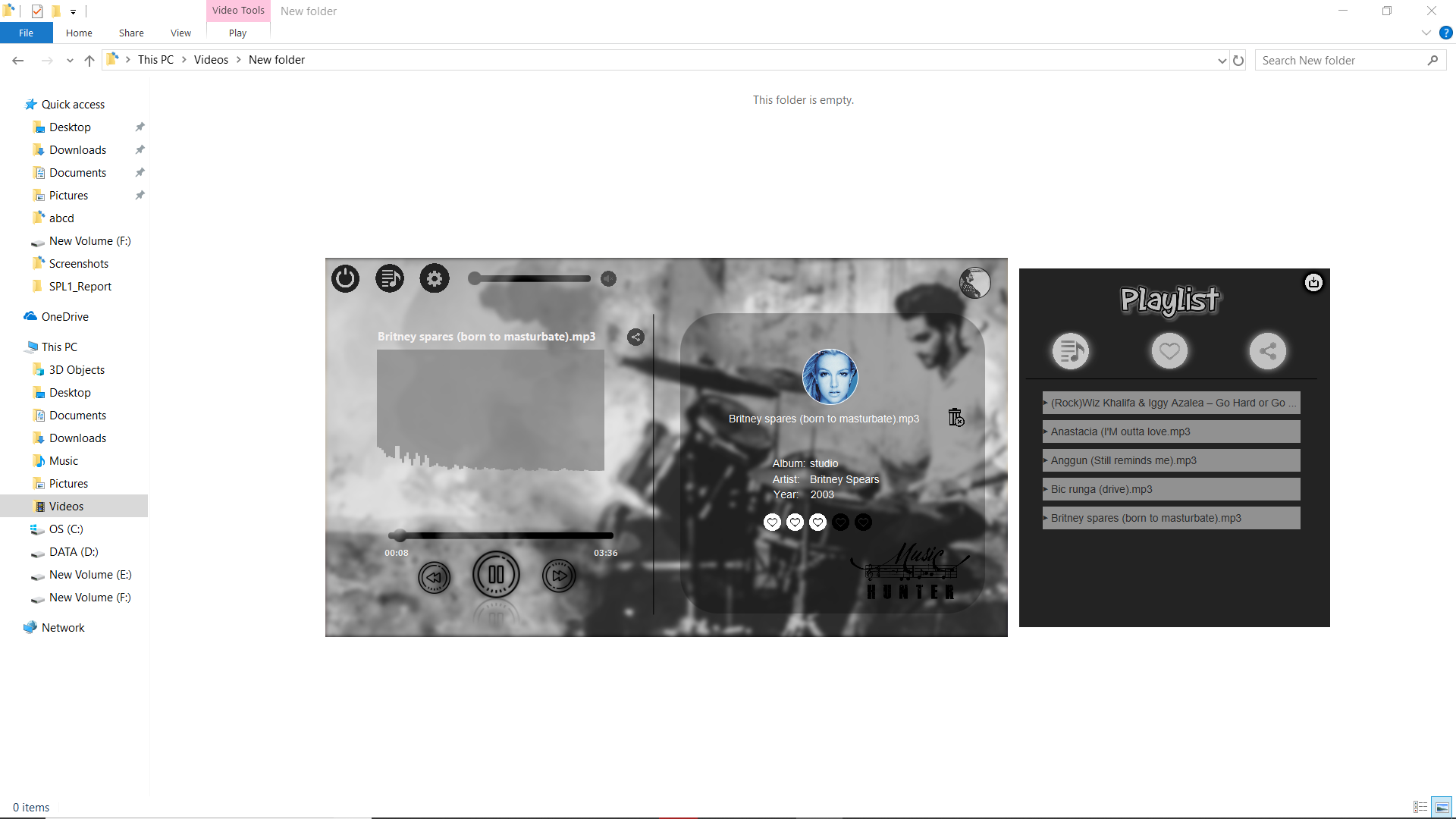


Figure (a)

**3.5.2 Automatically**

The program is continuously track the user state. If an user frequently listen a song then it might be a favorite song. So that if you listen a song 70% of total duration and not seeking the song or not set rating manually while playing the song it will assume that you listened it perfectly and if you do it repeatedly then the priority of that song will be increasing. After certain number of priority the rating of a song will be increasing and it will be automatically added into favorite list.

After certain period of time if user doesn’t listened that song then the song might be boring for the user. My program can also track this state. If user doesn’t listen it for a long period of time then the priority of song will be decreasing day by day and the rating will also be decreasing and when the rating will be ‘0’ then the song will be automatically removed from the user favorite list.

**3.6 Message Encryption and Decryption**

**3.6.1 Encryption**

To encrypt message make an object of **Encryption** class and give a source of an audio file which must be WAV format shown in figure (a).

After make an object call **injectData()** to push the message into audio data where the parameter of **injectData()** method is message which I want to encrypt. Message must be ends with full stop (.) .

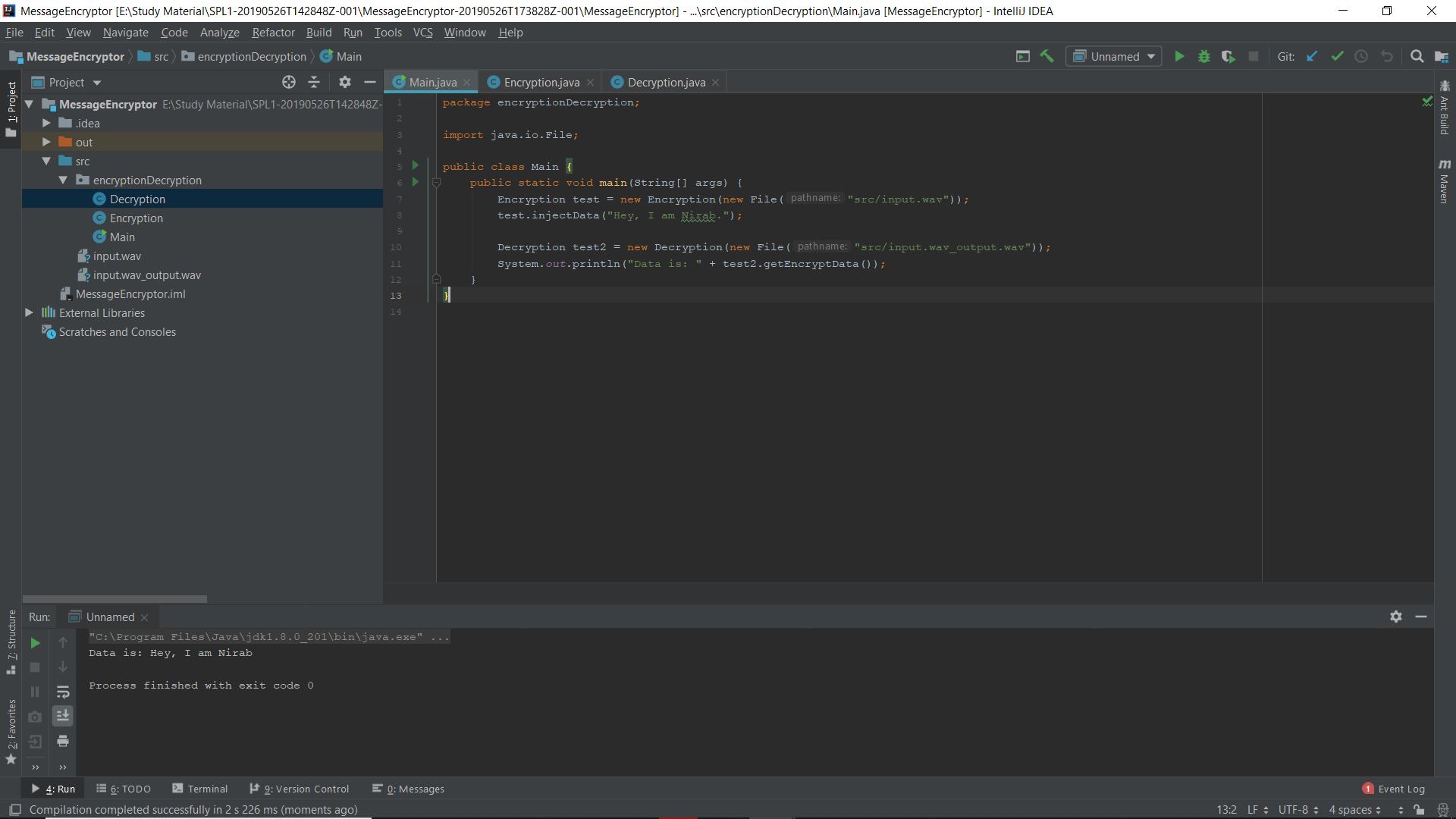


Figure (a)

**3.6.2 Decryption**

To decrypt message from an audio data you should make a Decryption class object and pass a source of an audio file which contains encrypted message.

And finally **getEncryptedData()** returns a decrypted message which is shown in figure (a)

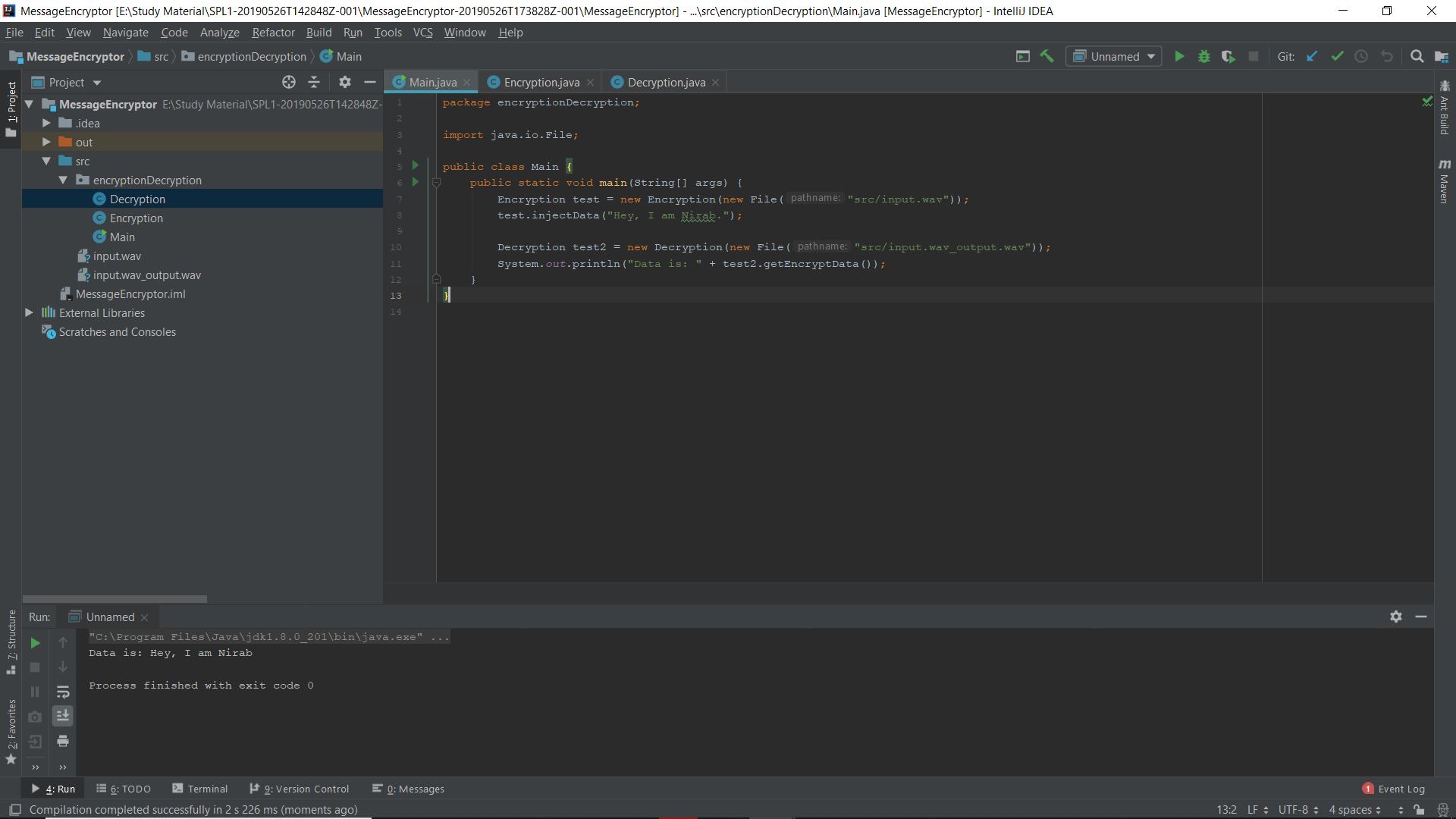


Figure (a)

**4. Conclusion**

Implementing SHA-1 and Gesture recignation helps me to understand this new topic and how to implement it in real life problem. The project also helps me to improve my coding skill and I have learned to handle large code for the first time. I have learnt to use java in different scenario. I hope it will help me to deal with difficulties in future. This project was quiet challenging and I gained a lot of experience from it. I want to thank my supervisor and other respectable teachers for guiding me a lot during this project.

**And to make this project I have to write 4,694 lines of code without GUI. With GUI the number of lines are 5,456**

**5. Appendix**

In future I’ll developed my project and add some interesting features which is eye catching.

**Thank you**

**Reference**

1. <https://en.wikipedia.org/wiki/SHA-1> [20-04-2019]
2. <https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm> [17-03-2019]
3. <http://www.topherlee.com/software/pcm-tut-wavformat.html> [28-05-2019]