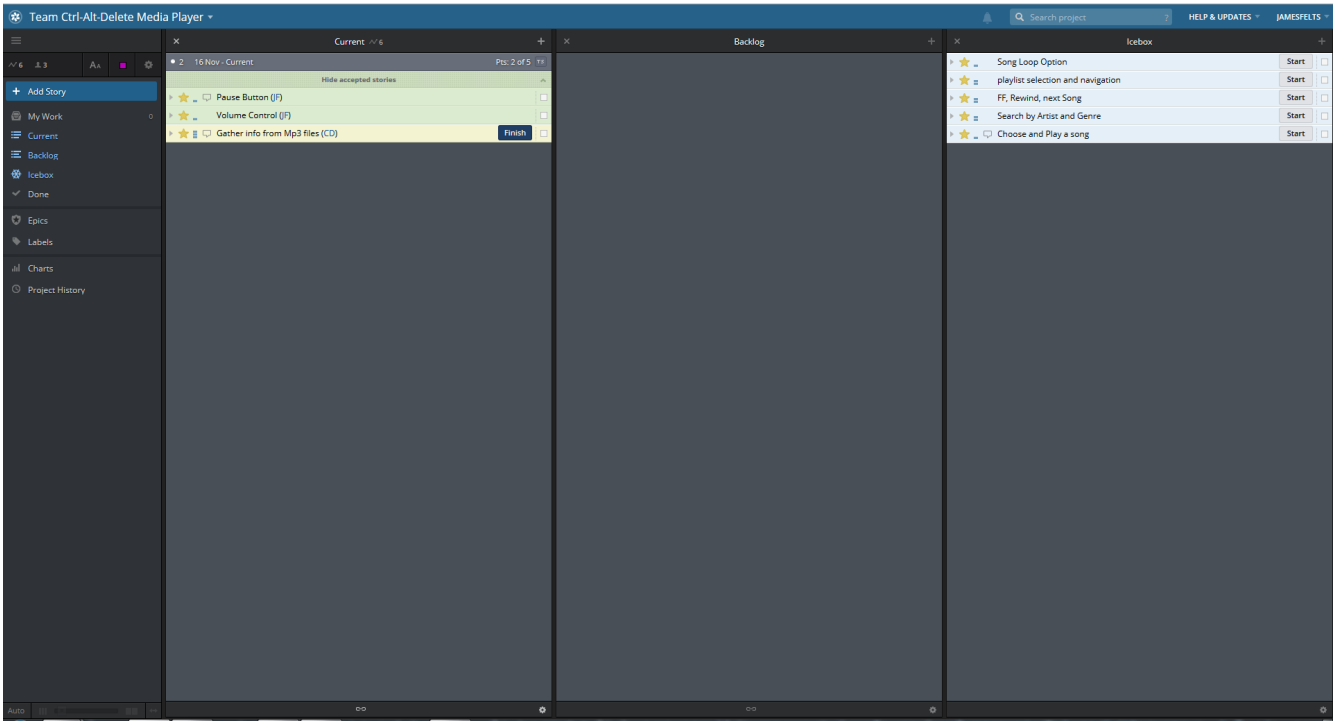
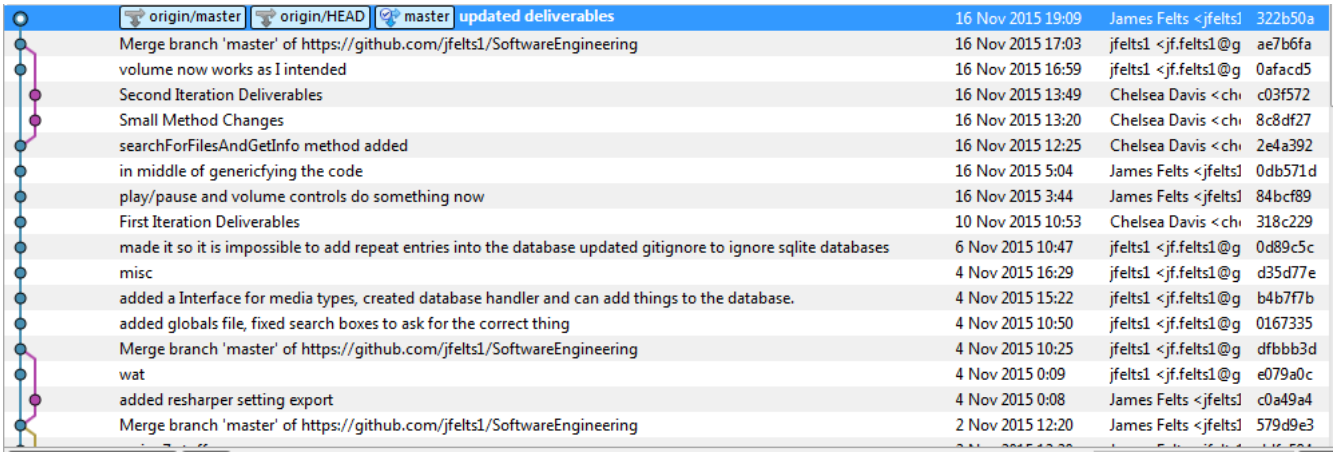
Team Ctrl-Alt-Delete
Second Iteration Deliverables

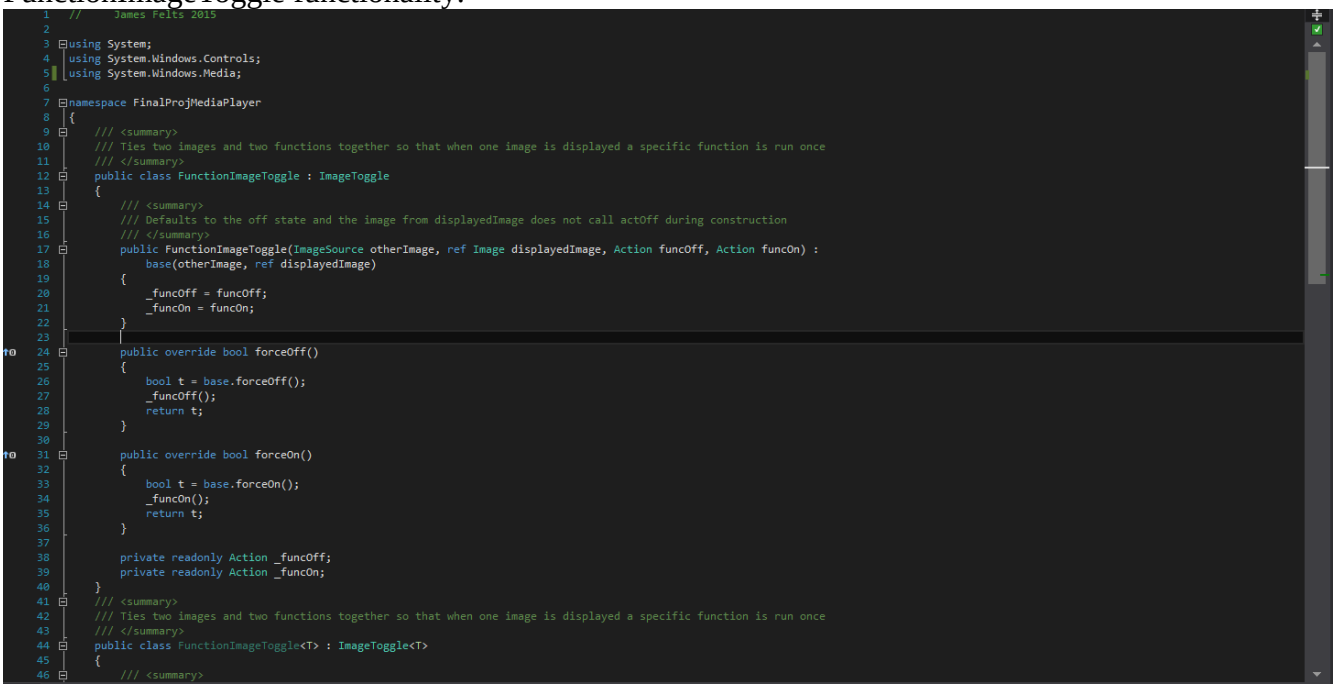
Pivotal Tracker:



Git/SourceTree:

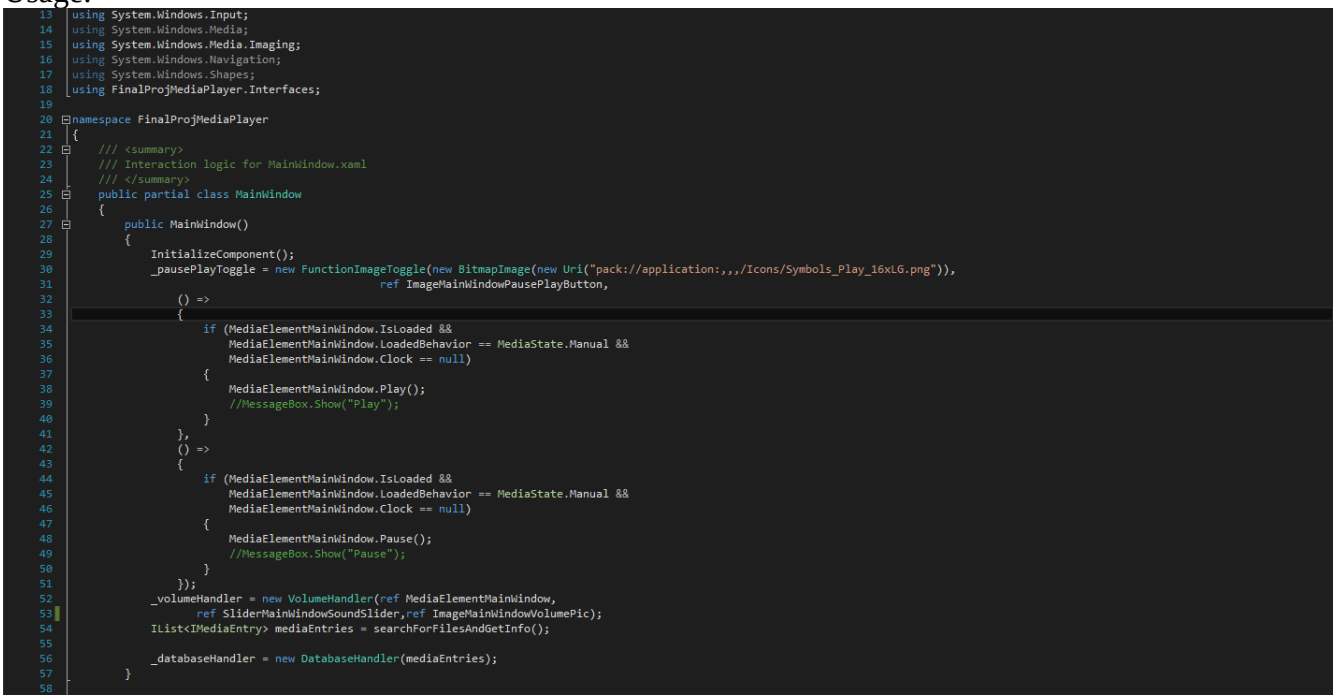


FunctionImageToggle functionality:



```
//     James Fells 2015

using System;
using System.Windows.Controls;
using System.Windows.Media;

namespace FinalProjMediaPlayer
{
    /// <summary>
    /// Ties two images and two functions together so that when one image is displayed a specific function is run once
    /// </summary>
    public class FunctionImageToggle : ImageToggle
    {
        /// <summary>
        /// Defaults to the off state and the image from displayedImage does not call actOff during construction
        /// </summary>
        public FunctionImageToggle(ImageSource otherImage, ref Image displayedImage, Action funcOff, Action funcOn) :
            base(otherImage, ref displayedImage)
        {
            _funcOff = funcOff;
            _funcOn = funcOn;
        }

        public override bool forceOff()
        {
            bool t = base.forceOff();
            _funcOff();
            return t;
        }

        public override bool forceOn()
        {
            bool t = base.forceOn();
            _funcOn();
            return t;
        }

        private readonly Action _funcOff;
        private readonly Action _funcOn;
    }
    /// <summary>
    /// Ties two images and two functions together so that when one image is displayed a specific function is run once
    /// </summary>
    public class FunctionImageToggle<T> : ImageToggle<T>
    {
        /// <summary>
```

Usage:



```
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using FinalProjMediaPlayer.Interfaces;

namespace FinalProjMediaPlayer
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow
    {
        public MainWindow()
        {
            InitializeComponent();
            _pausePlayToggle = new FunctionImageToggle(new BitmapImage(new Uri("pack://application:,,,/Icons/Symbols_Play_16xLG.png")),
                                                       ref ImageMainWindowPausePlayButton,
                () =>
                {
                    if (MediaElementMainWindow.IsLoaded &&
                        MediaElementMainWindow.LoadedBehavior == MediaState.Manual &&
                        MediaElementMainWindow.Clock == null)
                    {
                        MediaElementMainWindow.Play();
                        //MessageBox.Show("Play");
                    }
                },
                () =>
                {
                    if (MediaElementMainWindow.IsLoaded &&
                        MediaElementMainWindow.LoadedBehavior == MediaState.Manual &&
                        MediaElementMainWindow.Clock == null)
                    {
                        MediaElementMainWindow.Pause();
                        //MessageBox.Show("Pause");
                    }
                });
            _volumeHandler = new VolumeHandler(ref MediaElementMainWindow,
                  ref SliderMainWindowSoundSlider,ref ImageMainWindowVolumePic);
            IList<IMediaEntry> mediaEntries = searchForFilesAndGetInfo();

            _databaseHandler = new DatabaseHandler(mediaEntries);
        }
```

VolumeHandler:

```
namespace FinalProjMediaPlayer
{
    public class VolumeHandler:IToggle
    {
        public VolumeHandler(ref MediaElement mediaElement, ref Slider volumeSlider,ref Image pic)
        {
            _mediaElement = mediaElement;
            _volumeImageToggle = new ImageToggle(
                new BitmapImage(new Uri("pack://application:,,,/Icons/SoundfileNoSound_461.png")),ref pic);
            _lastVolumeValue = Globals.MaxVolume;
            _lastVolumeSliderValue = Globals.MaxSliderValue;
            _volumeSlider = volumeSlider;
            forceOn();
        }

        public bool toggle()
        {
            return Toggled ? forceOff() : forceOn();
        }

        public bool forceOff()
        {
            Toggled = false;
            _mediaElement.Volume = 0;
            _volumeSlider.Value = 0;
            _volumeImageToggle.forceOn();
            return Toggled;
        }

        public bool forceOn()
        {
            Toggled = true;
            _mediaElement.Volume = _lastVolumeValue;
            _volumeSlider.Value = _lastVolumeSliderValue;
            _volumeImageToggle.forceOff();
            return Toggled;
        }

        public void setVolume(object sender, RoutedPropertyChangedEventArgs<double> e)
        {
            double tmp = e.NewValue / Globals.MaxSliderValue;

            if (!(tmp <= 1) || !(tmp >= 0))
            {
                MessageBox.Show("Invalid Volume: " + tmp);
            }
```

```
            _volumeSlider.Value = 0;
            _volumeImageToggle.forceOn();
            return Toggled;
        }

        public bool forceOn()
        {
            Toggled = true;
            _mediaElement.Volume = _lastVolumeValue;
            _volumeSlider.Value = _lastVolumeSliderValue;
            _volumeImageToggle.forceOff();
            return Toggled;
        }

        public void setVolume(object sender, RoutedPropertyChangedEventArgs<double> e)
        {
            double tmp = e.NewValue / Globals.MaxSliderValue;

            if (!(tmp <= 1) || !(tmp >= 0))
            {
                MessageBox.Show("Invalid Volume: " + tmp);
            }

            if (Math.Abs(e.NewValue) < Globals.DoubleTolerance)
            {
                forceOff();
            }
            else
            {
                _lastVolumeValue = tmp;
                _lastVolumeSliderValue = e.NewValue;
                forceOn();
            }
        }

        public bool Toggled { get; private set; }

        private readonly MediaElement _mediaElement;
        private readonly IToggle _volumeImageToggle;
        private readonly Slider _volumeSlider;
        private double _lastVolumeValue;
        private double _lastVolumeSliderValue;

    }
}
```

Search for media files:

```
61
62          private IList<IMediaEntry> searchForFilesAndGetInfo()
63          {
64              //Jess' code starts here
65              string[] mp3Files = System.IO.Directory.GetFiles(System.IO.Directory.GetCurrentDirectory(), "*.mp3");
66              string[] aviFiles = System.IO.Directory.GetFiles(System.IO.Directory.GetCurrentDirectory(), "*.avi");
67              Console.WriteLine("Current Working Directory: " + System.IO.Directory.GetCurrentDirectory());
68              ArrayList files = new ArrayList(mp3Files.Length + aviFiles.Length);
69              if (mp3Files.Length == 0) Console.WriteLine("Error: No_MP3_Doge"); // (ノ °□°) ノ ︵ ┻━┻
70              else
71              {
72                  foreach (string t in mp3Files)
73                  {
74                      Console.WriteLine(t);
75                      files.Add(t);
76                  }
77              }
78              if (aviFiles.Length == 0) Console.WriteLine("Error: No_AVI_Doge"); // (ノ °□°) ノ ︵ ┻━┻
79              else
80              {
81                  foreach (string t in aviFiles)
82                  {
83                      Console.WriteLine(t);
84                      files.Add(t);
85                  }
86              }
88              Console.WriteLine("ArrayList: ");
89              foreach (object curLine in files)
90              {
91                  Console.WriteLine(curLine);
92              }
93              Console.ReadLine();
94
```

```
            string filePath = "";
            IList<IMediaEntry> mediaEntries = new List<IMediaEntry>();

            if(files.Count == 0) Console.WriteLine("Error: No_Files");
            else
            {
                foreach (object filePathObject in files)
                {
                    filePath = filePathObject.ToString();

                    using (FileStream fs = File.OpenRead(filePath))
                    {
                        //Meaning that tags have been added to the file, ie artist, genre, etc
                        if (fs.Length >= 128)
                        {
                            MusicID3Tag tag = new MusicID3Tag();
                            fs.Seek(-128, SeekOrigin.End);
                            fs.Read(tag.TAGID, 0, tag.TAGID.Length);
                            fs.Read(tag.Title, 0, tag.Title.Length);
                            fs.Read(tag.Artist, 0, tag.Artist.Length);
                            fs.Read(tag.Album, 0, tag.Album.Length);
                            fs.Read(tag.Year, 0, tag.Year.Length);
                            fs.Read(tag.Comment, 0, tag.Comment.Length);
                            fs.Read(tag.Genre, 0, tag.Genre.Length);
                            string theTAGID = Encoding.Default.GetString(tag.TAGID);

                            if (theTAGID.Equals("TAG"))
                            {
                                string mediaTitle = Encoding.Default.GetString(tag.Title);
                                string mediaArtist = Encoding.Default.GetString(tag.Artist);
                                string mediaAlbum = Encoding.Default.GetString(tag.Album);
                                string mediaYear = Encoding.Default.GetString(tag.Year);
                                string mediaComment = Encoding.Default.GetString(tag.Comment);
                                string mediaGenre = Encoding.Default.GetString(tag.Genre);
                                //long Length = Encoding.Default.GetString(tag.)

                                Console.WriteLine();
                                Console.WriteLine("Title: " + mediaTitle);
                                Console.WriteLine("Artist: " + mediaArtist);
                                Console.WriteLine("Album: " + mediaAlbum);
                                Console.WriteLine("Year: " + mediaYear);
                                Console.WriteLine("Comment: " + mediaComment);
                                Console.WriteLine("Genre: " + mediaGenre);
                                Console.WriteLine();

                                // MusicEntry(string genre, string title, long length, string artist, string filePath)
                                MusicEntry m = new MusicEntry(mediaGenre, mediaTitle, 0, mediaArtist, filePath);
                                mediaEntries.Add(m);
                            }
                        }
                    }
                }

            }

            return mediaEntries;
        }
```