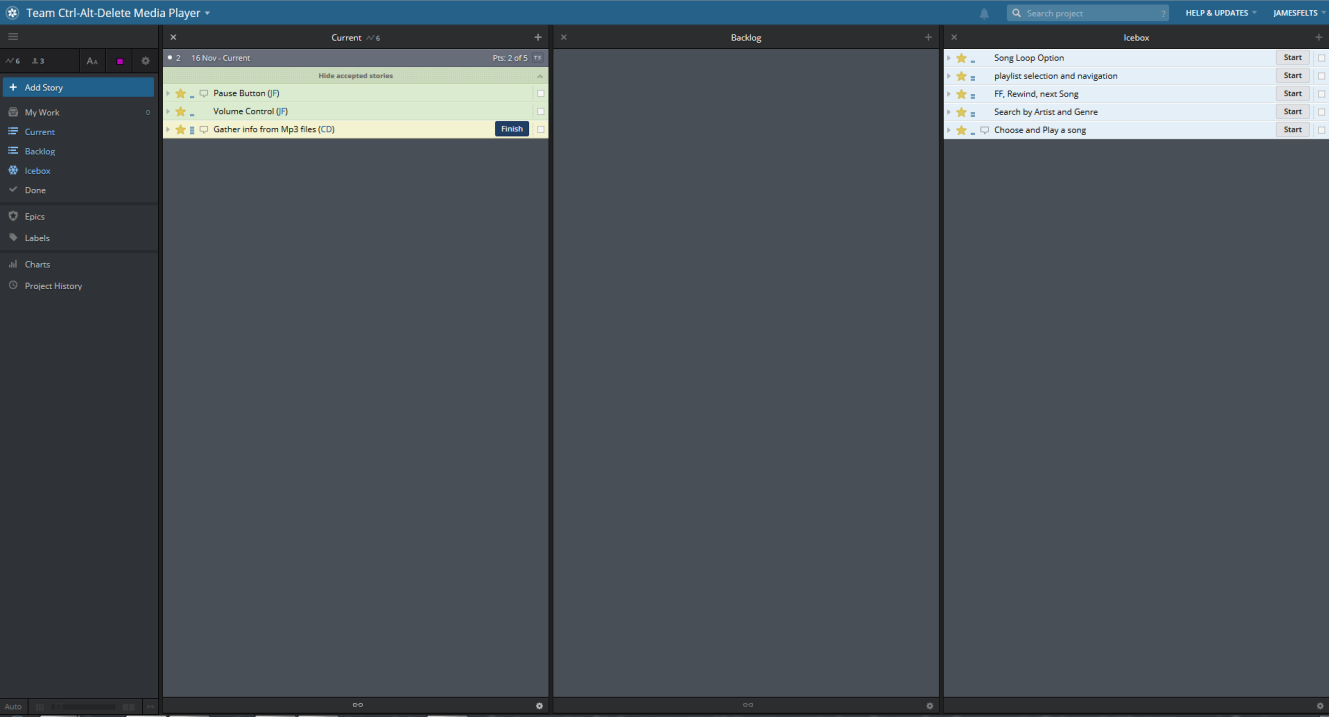


Team Ctrl-Alt-Delete
Second Iteration Deliverables

Pivotal Tracker:



Git/SourceTree:

	origin/master	origin/HEAD	master	updated deliverables	16 Nov 2015 19:09	James Felts <jfelts1	322b50a
				Merge branch 'master' of https://github.com/jfelts1/SoftwareEngineering	16 Nov 2015 17:03	jfelts1 <jf.felts1@g	ae7b6fa
				volume now works as I intended	16 Nov 2015 16:59	jfelts1 <jf.felts1@g	0afacd5
				Second Iteration Deliverables	16 Nov 2015 13:49	Chelsea Davis <chi	c03f572
				Small Method Changes	16 Nov 2015 13:20	Chelsea Davis <chi	8c8df27
				searchForFilesAndGetInfo method added	16 Nov 2015 12:25	Chelsea Davis <chi	2e4a392
				in middle of genericfyng the code	16 Nov 2015 5:04	James Felts <jfelts1	0db571d
				play/pause and volume controls do something now	16 Nov 2015 3:44	James Felts <jfelts1	84bcf89
				First Iteration Deliverables	10 Nov 2015 10:53	Chelsea Davis <chi	318c229
				made it so it is impossible to add repeat entries into the database updated gitignore to ignore sqlite databases	6 Nov 2015 10:47	jfelts1 <jf.felts1@g	0d89c5c
				misc	4 Nov 2015 16:29	jfelts1 <jf.felts1@g	d35d77e
				added a Interface for media types, created database handler and can add things to the database.	4 Nov 2015 15:22	jfelts1 <jf.felts1@g	b4b7f7b
				added globals file, fixed search boxes to ask for the correct thing	4 Nov 2015 10:50	jfelts1 <jf.felts1@g	0167335
				Merge branch 'master' of https://github.com/jfelts1/SoftwareEngineering	4 Nov 2015 10:25	jfelts1 <jf.felts1@g	dfbbb3d
				wat	4 Nov 2015 0:09	jfelts1 <jf.felts1@g	e079a0c
				added resharper setting export	4 Nov 2015 0:08	James Felts <jfelts1	c0a49a4
				Merge branch 'master' of https://github.com/jfelts1/SoftwareEngineering	2 Nov 2015 12:20	James Felts <jfelts1	579d9e3

FunctionImageToggle functionality:

```
1  // James Reits 2015
2
3  using System;
4  using System.Windows.Controls;
5  using System.Windows.Media;
6
7  namespace FinalProjMediaPlayer
8  {
9      /// <summary>
10     /// Ties two images and two functions together so that when one image is displayed a specific function is run once
11     /// </summary>
12     public class FunctionImageToggle : ImageToggle
13     {
14         /// <summary>
15         /// Defaults to the off state and the image from displayedImage does not call actOff during construction
16         /// </summary>
17         public FunctionImageToggle(ImageSource otherImage, ref Image displayedImage, Action funcOff, Action funcOn) :
18             base(otherImage, ref displayedImage)
19         {
20             _funcOff = funcOff;
21             _funcOn = funcOn;
22         }
23
24         public override bool forceOff()
25         {
26             bool t = base.forceOff();
27             _funcOff();
28             return t;
29         }
30
31         public override bool forceOn()
32         {
33             bool t = base.forceOn();
34             _funcOn();
35             return t;
36         }
37
38         private readonly Action _funcOff;
39         private readonly Action _funcOn;
40     }
41     /// <summary>
42     /// Ties two images and two functions together so that when one image is displayed a specific function is run once
43     /// </summary>
44     public class FunctionImageToggle<T> : ImageToggle<T>
45     {
46         /// <summary>
```

Usage:

```
13  using System.Windows.Input;
14  using System.Windows.Media;
15  using System.Windows.Media.Imaging;
16  using System.Windows.Navigation;
17  using System.Windows.Shapes;
18  using FinalProjMediaPlayer.Interfaces;
19
20  namespace FinalProjMediaPlayer
21  {
22      /// <summary>
23      /// Interaction logic for MainWindow.xaml
24      /// </summary>
25      public partial class MainWindow
26      {
27          public MainWindow()
28          {
29              InitializeComponent();
30              _pausePlayToggle = new FunctionImageToggle(new BitmapImage(new Uri("pack://application:,,,/Icons/Symbols_Play_16xLG.png")),
31                  ref ImageMainWindowPausePlayButton,
32                  () =>
33                  {
34                      if (MediaElementMainWindow.IsLoaded &&
35                          MediaElementMainWindow.LoadedBehavior == MediaState.Manual &&
36                          MediaElementMainWindow.Clock == null)
37                      {
38                          MediaElementMainWindow.Play();
39                          //MessageBox.Show("Play");
40                      }
41                  },
42                  () =>
43                  {
44                      if (MediaElementMainWindow.IsLoaded &&
45                          MediaElementMainWindow.LoadedBehavior == MediaState.Manual &&
46                          MediaElementMainWindow.Clock == null)
47                      {
48                          MediaElementMainWindow.Pause();
49                          //MessageBox.Show("Pause");
50                      }
51                  });
52              _volumeHandler = new VolumeHandler(ref MediaElementMainWindow,
53                  ref SliderMainWindowSoundSlider, ref ImageMainWindowVolumePic);
54              IList<IMediaEntry> mediaEntries = searchForFilesAndGetInfo();
55
56              _databaseHandler = new DatabaseHandler(mediaEntries);
57          }
58      }
```

VolumeHandler:

```
10 namespace FinsIPProj.MediaPlayer
11 {
12     public class VolumeHandler:IToggle
13     {
14         public VolumeHandler(ref MediaElement mediaElement, ref Slider volumeSlider, ref Image pic)
15         {
16             _mediaElement = mediaElement;
17             _volumeImageToggle = new ImageToggle(
18                 new BitmapImage(new Uri("pack://application:,,,/Icons/SoundfileNoSound_461.png")), ref pic);
19             _lastVolumeValue = Globals.MaxVolume;
20             _lastVolumeSliderValue = Globals.MaxSliderValue;
21             volumeSlider = volumeSlider;
22             forceOn();
23         }
24
25         public bool toggle()
26         {
27             return Toggled ? forceOff() : forceOn();
28         }
29
30         public bool forceOff()
31         {
32             Toggled = false;
33             _mediaElement.Volume = 0;
34             _volumeSlider.Value = 0;
35             _volumeImageToggle.forceOn();
36             return Toggled;
37         }
38
39         public bool forceOn()
40         {
41             Toggled = true;
42             _mediaElement.Volume = _lastVolumeValue;
43             _volumeSlider.Value = _lastVolumeSliderValue;
44             _volumeImageToggle.forceOff();
45             return Toggled;
46         }
47
48         public void setVolume(object sender, RoutedPropertyChangedEventArgs<double> e)
49         {
50             double tmp = e.NewValue / Globals.MaxSliderValue;
51             if (!(tmp <= 1) || !(tmp >= 0))
52             {
53                 MessageBox.Show("Invalid Volume: " + tmp);
54             }
55         }
56     }
57 }
```

```
33         _volumeSlider.Value = 0;
34         _volumeImageToggle.forceOn();
35         return Toggled;
36     }
37
38     public bool forceOn()
39     {
40         Toggled = true;
41         _mediaElement.Volume = _lastVolumeValue;
42         _volumeSlider.Value = _lastVolumeSliderValue;
43         _volumeImageToggle.forceOff();
44         return Toggled;
45     }
46
47     public void SetVolume(object sender, RoutedPropertyChangedEventArgs<double> e)
48     {
49         double tmp = e.NewValue / Globals.MaxSliderValue;
50         if (!(tmp <= 1) || !(tmp >= 0))
51         {
52             MessageBox.Show("Invalid Volume: " + tmp);
53         }
54
55         if (Math.Abs(e.NewValue) < Globals.DoubleTolerance)
56         {
57             forceOff();
58         }
59         else
60         {
61             _lastVolumeValue = tmp;
62             _lastVolumeSliderValue = e.NewValue;
63             forceOn();
64         }
65     }
66
67     public bool Toggled { get; private set; }
68
69     private readonly MediaElement _mediaElement;
70     private readonly IToggle _volumeImageToggle;
71     private readonly Slider _volumeSlider;
72     private double _lastVolumeValue;
73     private double _lastVolumeSliderValue;
74 }
75
76
77 }
```

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     }
87     Console.WriteLine("ArrayList: ");
88     foreach (object curLine in files)
89     {
90         Console.WriteLine(curLine);
91     }
92     Console.ReadLine();
93
94 }
```

```

98     string filePath = "";
99     IList<IMediaEntry> mediaEntries = new List<IMediaEntry>();
100
101     if(files.Count == 0) Console.WriteLine("Error: No_Files");
102     else
103     {
104         foreach (object filePathObject in files)
105         {
106             filePath = filePathObject.ToString();
107
108             using (FileStream fs = File.OpenRead(filePath))
109             {
110                 //Meaning that tags have been added to the file, ie artist, genre, etc
111                 if (fs.Length >= 128)
112                 {
113                     MusicID3Tag tag = new MusicID3Tag();
114                     fs.Seek(-128, SeekOrigin.End);
115                     fs.Read(tag.TAGID, 0, tag.TAGID.Length);
116                     fs.Read(tag.Title, 0, tag.Title.Length);
117                     fs.Read(tag.Artist, 0, tag.Artist.Length);
118                     fs.Read(tag.Album, 0, tag.Album.Length);
119                     fs.Read(tag.Year, 0, tag.Year.Length);
120                     fs.Read(tag.Comment, 0, tag.Comment.Length);
121                     fs.Read(tag.Genre, 0, tag.Genre.Length);
122                     string theTAGID = Encoding.Default.GetString(tag.TAGID);
123
124                     if (theTAGID.Equals("TAG"))
125                     {
126                         string mediaTitle = Encoding.Default.GetString(tag.Title);
127                         string mediaArtist = Encoding.Default.GetString(tag.Artist);
128                         string mediaAlbum = Encoding.Default.GetString(tag.Album);
129                         string mediaYear = Encoding.Default.GetString(tag.Year);
130                         string mediaComment = Encoding.Default.GetString(tag.Comment);
131                         string mediaGenre = Encoding.Default.GetString(tag.Genre);
132                         //long Length = Encoding.Default.GetString(tag.)
133
134                         Console.WriteLine();
135                         Console.WriteLine("Title: " + mediaTitle);
136                         Console.WriteLine("Artist: " + mediaArtist);
137                         Console.WriteLine("Album: " + mediaAlbum);
138                         Console.WriteLine("Year: " + mediaYear);
139                         Console.WriteLine("Comment: " + mediaComment);
140                         Console.WriteLine("Genre: " + mediaGenre);
141                         Console.WriteLine();
142
143                         // MusicEntry(string genre, string title, long length, string artist, string filePath)
144                         MusicEntry m = new MusicEntry(mediaGenre, mediaTitle, 0, mediaArtist, filePath);
145                         mediaEntries.Add(m);
146                     }
147                 }
148             }
149         }
150     }
151
152     return mediaEntries;
153 }
154

```